PT2001SWUG

PT2001 programming guide and instruction set

Rev. 3.0 — 29 April 2019 User guide

1 Introduction

This programming guide relates to the PT2001 Programmable Solenoid Controller. Refer to the individual device data sheet for feature information. The PT2001 programming guide describes the microcore programming model, instruction set, data types used, and basic memory organization. Also included in the guide is an example of microcode used in the FRDMPT2001EVM using most of the instructions described herein. The programming of the microcores has to be done using the PT2001 IDE, which is available at https://www.nxp.com.

2 Microcore programming description

2.1 CRAM addressing mode

All the jump instructions have two possible outcomes: if a specific condition (if any) is true, then the code flow continues at a destination specified by a parameter, otherwise it continues to the next code line. In the same way, when a wait entry is configured, a parameter specifies the destination.

The instruction set of PT2001 allows only two addressing modes to express the destination parameter for the CRAM:

- **Relative address** (relative): The relative address parameter is represented by 5 bits. The physical address of the destination is obtained by adding the relative address to the physical address of the instruction that uses the parameter (that is the value of the uprogram counter when the instruction is executed). The relative address must be considered as two's complement represented and must be extended on 10 bit before the addition. By using relative addresses it is possible to range from "current_address – 16" to "current_address + 15".
- **Indirect address** (far)**:** It is possible to jump to the CRAM address contained into one of two jump_registers (jr1 or jr2). These registers can be loaded with a dedicated instruction and simply referred to in the wait or jump instructions. Refer to ldjr1 and ldjr2 in Section 3.2 "Instruction set".

2.2 Arithmetic logic unit

The microcore contains a simple Arithmetic Logic Unit (ALU). The ALU has an 8-word internal register file connected to the internal bus. The ALU can perform the following operations:

- **Addition and subtraction.** These operations are completed in a single ck clock cycle.
- **Multiplication.** This operation is completed in up to 32 ck clock cycles. The result is available as a 32-bit number, and is always in the registers GPR6 (MSBs) and GPR7 (LSBs).

- **Shift operations.** The operand is shifted one position (left or right) each ck clock cycle, so it requires from 1 to 16 ck clock cycles to execute. The shift operations always consume the operand. It is also possible to shift an operand by eight positions (left or right) or to swap the eight MSBs with the eight LSBs in one ck clock cycle.
- **Logic operation.** It is possible to operate a bitwise logical operation (and, not, or, xor) between an operand and a mask. It is also possible to bitwise invert an operand. All these operations are completed in a single ck clock cycle. These operations always consume the operand.

C2 conversions. It is possible to convert data from an unsigned representation to two's complement and vice versa. These operations are completed in a single ck clock cycle.

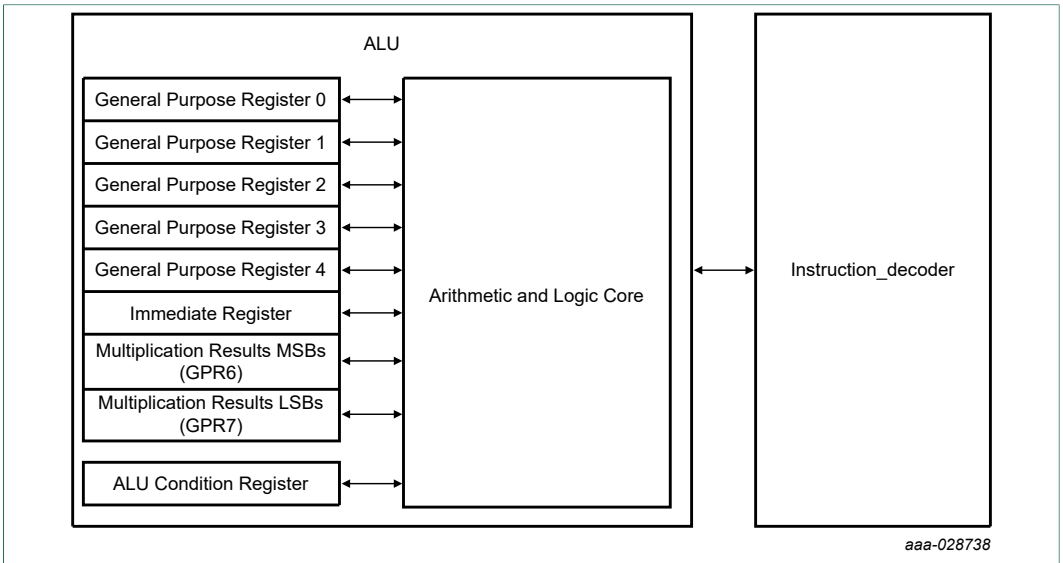

Figure 1. ALU block diagram

These operations consume the operand. While the ALU is busy performing an operation, request of other operations is impossible. In such cases, the request is ignored by the ALU.

The ALU instructions are:

- Addition (add), addition with immediate (addi)
- Subtraction (sub), subtraction with immediate (subi)
- Multiplication (mul), multiplication with immediate (muli)
- Logical operation (and, not, or, xor)
- Conversion from positive to two's complement (toc2) and from two's complement to positive (toint)
- Shift operation (sh32r, sh32l, shl, shr, shls, shrs), shift operation with immediate (sh32ri, sh32li, shli, shri, shlsi, shrsi), and byte manipulation shift (shl8, shr8, swap)
- ALU configuration (stal)

Some ALU instructions are multicycle (mul, muli and possibly sh32r, sh32l, shl, shr, shls, shrs, sh32ri, sh32li, shli, shri, shlsi, shrsi, depending on how many shift positions are required). While a multicycle operation is in progress, all ALU instructions are ignored, except for the stal instruction.

During this time any operations which try to modify the ALU registers (GPR0-7, arith_reg) are ignored (ldirl, ldirh and possibly cp, load if their destination address is one of the ALU registers). Instructions which try to read the ALU registers are successful (possibly cp

and store). It is possible to transfer constant values to the ALU immediate register using the ldirl and ldirh instructions.

When a multicycle instruction is required, it is recommended to wait until the operation is finished (ex: cwer Dest opd row1) before going to the next instruction. The operation completion can be checked by reading back the bit OP_DONE of the Arithmetic Condition Register (arith_reg).

2.2.1 Arithmetic condition register

The 16-bit register contain the status of the ALU concerning the last requested operation.

Table 1. Arithmetic condition register

Bit	Name	Description
15	SHIFT_OUT	Shifted out bit
14	CONV_SIGN	Last conversion sign
13	CARRY	Carryover bit
12 to 1	ARITH_LOGIC	Arithmetic logic
10	MASK_MIN	Mask result 0000h
9	MASK_MAX	Mask result FFFFh
8	MUL_SHIFT_OVR	Multiplication shift overflow
7	MUL_SHIFT_LOSS	Multiplication shift precision loss
6	RES_ZERO	Addition or subtraction result is zero
5	RES_SIGN	Addition or subtraction sign result
4	UNSIGNED_UND	Unsigned underflow
3	UNSIGNED_OVR	Unsigned overflow
2	SIGNED_UND	Signed underflow
1	SIGNED_OVR	Signed overflow
0	OP_DONE	Operation complete

- SHIFT_OUT is the last bit shifted out (either left or right) from a shift operation.
- CONV_SIGN is the product of all signs removed by toint instruction. This bit can be reset by performing a toint conversion with an rst parameter.
- CARRY is the carry produced by the last addition or subtraction operation performed.
- ARITH_LOGIC is a parameter used for addition and subtraction operations. It has four possible values:
  - "00" or "10": no limitation is imposed on addition or subtraction. In case of an overflow, the result should be represented by 17 bits, but only the 16 LSBs of this result are put in the target register. In case of an underflow, the result put in the target register is "65536 - the correct result", which can always be represented on 16 bits.
  - "01": the result of addition or subtraction are saturated between the maximum possible value (if overflow) or the minimum possible value (if underflow). The numbers are considered to be two's complement representation, so they are saturated between 8000h (–32768) and 7FFFh (+32767).
- "11": the result of addition or subtraction are saturated between the maximum possible value (if overflow) or the minimum possible value (if underflow). The numbers are

considered to be unsigned, so they are saturated between FFFFh (65535) and 0000h (0).

- MASK_MAX bit is set if the result of the last mask operation is FFFFh.
- MASK_MIN bit is set if the result of the last mask operation is 0000h.
- MUL_SHIFT_OVR is set to 0 if the 16 MSBs of the last multiplication or 32-bit shift result are all 0, otherwise it is 1.
- MUL_SHIFT_LOSS is set to 0 if the 16 LSBs of the last multiplication or 32-bit shift result are all 0, otherwise it is 1.
- RES_ZERO is set if the result of the last addition or subtraction is zero.
- RES_SIGN is set if the result of the last addition or subtraction is negative.
- UNSIGNED_UND is set if the last addition or subtraction produced underflow, considering the operands as unsigned numbers.
- SIGNED_UND is set if the last addition or subtraction produced underflow, considering the operands as two's complement numbers.
- UNSIGNED_OVR is set if the last addition or subtraction produced overflow, considering the operands as unsigned numbers.
- SIGNED_OVR is set if the last addition or subtraction produced overflow, considering the operands as two's complement numbers.

The OP_DONE can be set by the ALU, and the Instruction_decoder can only read it. This bit is set to 0 when a multicycle operation is in progress, otherwise is set to 1. If an ALU operation is issued when another operation is in progress (that is when the OP_DONE is set to 0), the request is ignored.

2.3 Start management

The start management block is designed to provide an anti-glitch functionality in order to reject glitches on the input start signal and also to provide the gen_start_uc0, gen_start_uc1, start_latch_uc0 and start_latch_uc1 signals. The main purpose of this block is to generate the internal gen_start signals feeding the microcores starting from the startx pins. Each microcore can be sensitive to the 6 startx pins according to the sensitivity map defined in the register start_config_reg (104h, 124h).

This block also provides the start_latch_ucx signals; these 8-bit signals (1 for each microcore) are used by the corresponding microcore to check which startx pin was active when the currently ongoing actuation began. In this way each microcore can be configured to be sensitive for up to all the 6 startx pins. While the actuation is ongoing, it also has the ability to check the level of the startx pins in two different modes that can be selected. The gen_start_ucx and start_latch_ucx can be generated according to two different strategies. The strategies for the two signals can be separately selected in the start_config_reg (104h, 124h).

Transparent mode: The gen_start_ucx is high if at least one of the starx signals is high for which the corresponding microcore is sensitive (refer to register start_config_reg (104h, 124h)). The start_latch_ucx signal is a living copy of the 6 startx pins for which the channel can be sensitive.

Smart Latch mode: When a startx pin (to which the microcore is sensitive) goes high and the start_latch_ucx is "00000000", the gen_start_ucx is set and the current startx pin status is latched in the start_latch_ucx register. If a rising edge is detected on any other startx pin, it is ignored. The gen_start_ucx signal goes to 0 only when the startx pin initially latched goes low. The start_latch_ucx register is reset only by the microcode (and this is done usually when the actuation currently ongoing is stopped by the gen_start_ucx

falling edge). The gen_start_ucx signal does not go high, until the start_latch_ucx register has been reset.

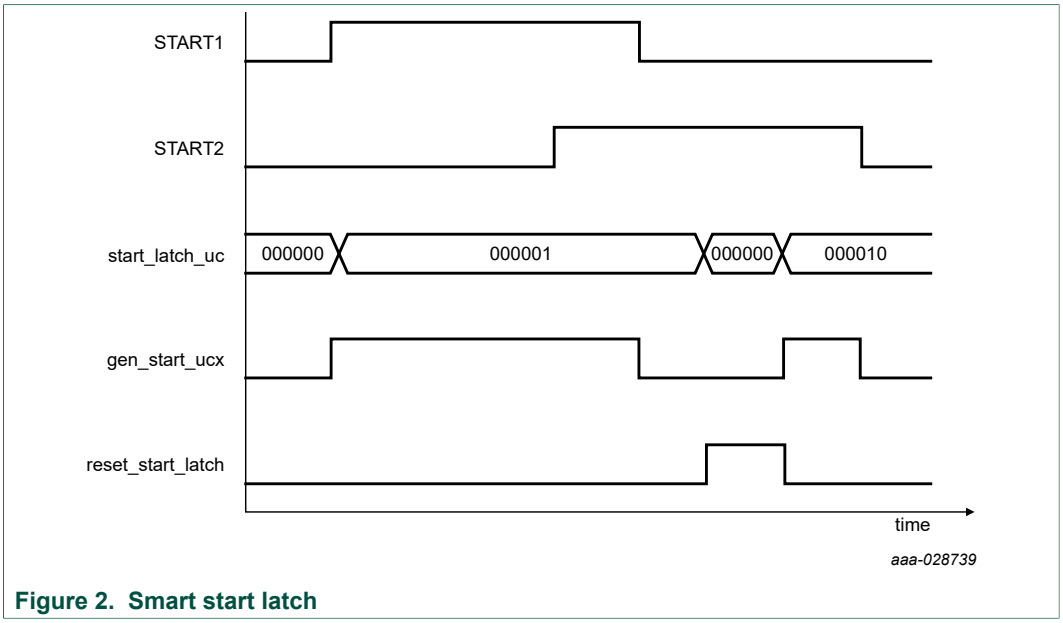

Figure 2. Smart start latch

The gen_start_ucx signals generated by this anti-glitch circuit are then also provided as an input to the corresponding microcores. If Smart Latch mode is enabled, no start edge is latched before the channel is locked by the flash enable bit.

2.4 Microprogram counter block

This block contains two registers: the microprogram counter (uPC) and the auxiliary register.

2.4.1 MicroPC

This is a 10-bit counter used to address the Code RAM containing the microprogram.

After the Code RAM is locked, this counter is loaded with an entry point selected through a SPI register (refer to the Uc0_entry_point (10Ah and 12Ah) and the Uc1_entry_point (10Bh and 12Bh) sections), which is the address of the first active instruction.

If an interrupt is requested, the uPC counter is moved to the appropriate interrupt routine register, as programmed in the parameter registers (refer to Diag_routine_addr (10Ch and 12Ch) and the Driver_disabled_routine_addr (10Dh, and 12Dh) and the Sw_interrupt_routine_addr (10Eh and 12Eh) sections). Only one level of interrupt is supported.

Before entering an interrupt routine, the interrupt status register is latched (refer to the Uc0_irq_status (10Fh, and 12Fh)). When an iret (interrupt return) instruction is executed, the interrupt status register is cleared and the uPC counter can be restored to the original address.

The instruction_decoder block directly controls the uPC in order to allow an efficient management of:

- Direct jumps
- Conditional jumps

- Subroutine execution
- Wait states

2.4.2 Auxiliary register

This 10-bit register is used to manage the one-level subroutine return or as an auxiliary memory element.

Any time the system executes a "jump to subroutine" instruction, the uPC is automatically stored in the auxiliary register before jumping to the subroutine start address. When the subroutine execution ends, the incremented auxiliary register content is transferred back to the uPC.

2.5 Wait instructions

The PT2001 is an event/response machine. An event occurs and then code executes, the wait instructions are the key to this behavior. The core waits at a 'wait' instruction for an event to occur.

These pending events are configured as rows in a six-row wait table. Before the wait instruction is issued, the wait table has to be configured with the cwef and cwer instructions to obtain the desired behavior. One instruction is required for each wait entry to be configured.

Although there are many possible event sources which can be configured inside the six row wait table:

- terminal_counts: any of the four terminal count (tc1, tc2, tc3, and tc4) signals can be checked to detect if any of the four counters has reached its end-of-count position.
- Flags: checks the value (both polarities) of one of the 16 flag signals available.
- Shortcut feedback: the voltage feedback (both polarities) related to the three shortcut outputs.
- gen_start: checks the value (both polarities) of the filtered chx_start input signal to define when to start and finish an actuation.
- current_feedback: the value (both polarities) of the six current feedbacks.
- own_current_feedback: the value (both polarities) of the own current feedbacks. This feedback can be different for each microcore and can be changed with the microcode instruction dfcsct. Table 2 "Current feedback assignment" shows the configuration after reset.

Table 2. Current feedback assignment

Microcore	Own current feedback (reset value)
Uc0,channel 1	current feedback 1
Uc1,channel 1	current feedback 2
Uc0,channel 2	current feedback 3
Uc1,channel 2	current feedback 4

- vboost: the output (both polarities) of the comparator that measures the boost voltage.

op_done: check if a previously issued ALU operation is still in progress or it is completed. This is mandatory for multiple cycle instructions (like mul, muli and possibly sh32r, sh32l, shl, shr, shls, shrs, sh32ri, sh32li, shli, shri, shlsi, shrsi, depending on how many shift positions are required)

Table 3. Wait instructions

cwef	Create wait table entry far
cwer	Create wait table entry relative
wait	Wait until condition satisfied

2.6 Subroutine instructions

This section covers the instructions that support calling and returning from subroutines. As explained in the CRAM addressing mode, the jump to subroutine can be relative if the destination address is in a range from "current_address - 16" to "current_address +15". If not, instruction jump far to subroutine needs to be used. When a subroutine instruction is set, the program counter (pc) is saved in the auxiliary register 'aux'. The rfs instruction "return from subroutine" causes the program counter to jump back to the main program.

Table 4. Subroutine instructions

jsrf	Jump far to subroutine
jtsr	Jump relative to subroutine
rfs	Return from subroutine

2.7 Program flow (jump, Ldjr) instructions

Conditions to be checked by the jump instructions are the same as the wait instruction with the addition of the following inputs:

- ctrl_reg: checks the value (both polarities) of one of the 16 control bits available in the ctrl_reg register (see register 101h, 102h, 121h,122h).
- status_bits: checks the value (both polarities) of one of the 16 control bits available in the Status_bits register (see register 105h, 106h, 125h, 126h).
- voltage feedback: the voltage feedback (both polarities) related to all the outputs.
- start_latch: checks the value of the six bit start_latch.
- arithmetic_register: checks the value (high polarity only) of one of the bits of the ALU arithmetic register. See Section 2.2 "Arithmetic logic unit".
- microcore_id: check if the microcore currently executing is uc0 or uc1

Same as the wait table, it is possible to jump far or jump relative. The following instructions need to be used to define the destination address when a jump far is required.

Table 5. Load jump registers instructions

ldjr1	Load jump register 1
ldjr2	Load jump register 2

Table 6 defines the list of different jump instructions which are triggered.

Table 6. Jump instructions

jarf	Jump far on arithmetic condition
jarr	Jump relative on arithmetic condition
jcrf	Jump far on control register condition
jcrr	Jump relative on control register condition

jfbkf	Jump far on feedback condition
jfbkr	Jump relative on feedback condition
jmpf	Unconditional jump far
jmpr	Unconditional jump relative
jocf	Jump far on condition
jocr	Jump relative on condition
joidf	Jump far on microcore condition
joidr	Jump relative on microcore condition
joslf	Jump far on start condition
joslr	Jump relative on start condition
jsrf	Jump far on status register bit condition
jsrr	Jump relative on status register bit condition
jtsf	Jump far to subroutine
jtsr	Jump relative to subroutine

2.8 DataRAM access instructions

The Data RAM access instructions are used to load and store data memory. These instructions also set the access mode which can be set to either 'Immediate' (_ofs parameter to be used) mode or 'Indexed' mode using the slab instruction. 'Indexed' mode is when an offset from the Base Address register is applied to the access's address (ofs parameter to be used). It is possible to modify the value of add_base with the stab instruction.

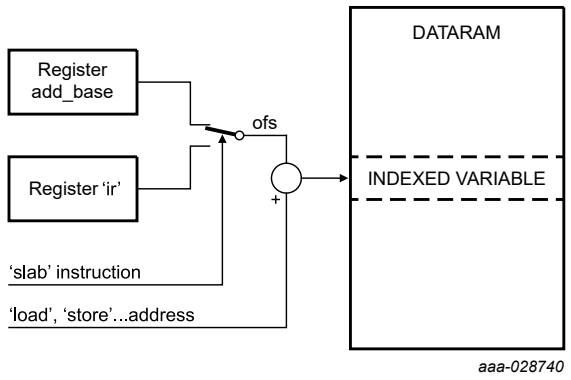

Figure 3. Indexed addressing mode

The three basic operations are:

- **Copy**. This instruction copies the value of one of the internal registers to another. The value of addr_base is neglected.
- **Load**. This instruction copies the value of a Data RAM element into one of the internal registers. A boolean parameter specifies if addr_base must be considered while addressing the Data RAM only.
- **Store**. This instruction copies the value of one of the internal registers to a Data RAM element. A boolean parameter specifies if addr_base must be considered while addressing the Data RAM only.

Table 7. DRAM access instructions

cp	Copy source register data to destination register
ldca	Load counter from ALU register and set outputs
ldcd	Load counter from Data RAM and set outputs
ldirh	Load 8-MSB ir register
ldirl	Load 8-LSB ir register
load	Load data from Data RAM to register
slab	Select Data RAM address base
stab	Set Data RAM address base
stdm	Set Data RAM read mode
store	Store register data in Data RAM

2.9 Arithmetic instructions

The Arithmetic Logic Unit (ALU) does math calculations and bitwise operations. The immediate register (ir) is used for most instructions. This register can be loaded using the ldrh and ldirl instructions.

Table 8. Math instructions

add	Add two ALU registers and place the result in one of the ALU registers
addi	Add an ALU register to the value in the immediate register and place the result in an ALU register
mul	Multiply two ALU registers and place the result in reg32
muli	Multiply an ALU register with the value in the immediate register and place the result in reg32
stal	Set arithmetic logic mode
sub	Subtract two ALU registers and place the result in one of the ALU registers
subi	Subtract the value in the immediate register from an ALU register and place the result in an ALU register
swap	Swap bytes inside ALU register
toc2	Convert an integer in an ALU register to 2's compliment format
toint	Convert the two's complement value contained in an ALU register to integer format

Table 9. Bitwise instructions

and	AND an ALU register with the value in the immediate register and place the result in the ALU register
not	Invert ALU register bits
or	OR an ALU register with the value in the immediate register and place the result in the ALU register
xor	XOR an ALU register with the value in the immediate register and place the result in the ALU register

2.10 Shift instructions

This section covers the shift instructions. Shifts include 'shift left' and 'shift right', 'shift by register' and 'shift immediate', 'normal shift' and 'signed shift' in which the most significant bit does not change, and 32-bit shifts in which the 'mh' and 'ml' registers are treated as a single 32-bit register in which the 'mh' register's lsb connects with the 'ml's registers msb.

Shifts take one instruction cycle per shifted bit and the 'arith_reg' register's 'OD' bit can be tested to determine when the shift is completed. So an 11-bit shift would normally take 11 clock cycles to execute. However, there is a special 8-bit shift which takes just a single clock cycle so shifts by constants greater than 8 bit positions can be accelerated by combining the 8-bit shift with the immediate shift.

Table 10. Shift instructions

sh32l	Shift left multiplication result register
sh32li	Shift left multiplication result register by immediate value
sh32r	Shift right multiplication result register
sh32ri	Shift right multiplication result register by immediate value
shl	Shift left ALU register
shl8	Shift left ALU register by 8 bits
shli	Shift left the ALU register by immediate value
shls	Shift left signed ALU register
shlsi	Shift left signed ALU register by immediate value
shr	Shift right ALU register
shr8	Shift right ALU register by 8 bits
shri	Shift right the ALU register immediate value
shrs	Shift right signed ALU register
shrsi	Shift right signed ALU register immediate value

2.11 Control, status, and flags instructions

This section covers the instructions that handle the control register, the status register and the flags register. Note that each of the four cores has its own control and status register but the four cores share the flag register.

The flags register has many purposes. The devices' input pins can be read through the (single) flags register (refer to register 1C1h and 1C3h). On the other hand, if flags are configured as output pins (refer to register 1C1h and 1C3h) they can be controlled through the flags register. The flags register can also be used by the 'wait' instruction to execute a section of code depending on its value high or low.

Table 11. Control, status and flags instructions

rstreg	Reset registers (control, status, automatic diagnostics...)
stcrb	Set control register bit
stf	Set flag
stsrb	Set status register bit

2.12 Intercore communication instructions

The intercore communication register 'rxtx' provides a mechanism to share data between cores. It is possible to exchange 16-bit data between different microcores, even belonging to different channels, using the ch_rxtx address in the internal memory map. Table 12 "ch_rxtx internal register in write mode" shows the register in write mode. The transmitting microcores can write data at this address; the receiving microcores can read the data using the same address, selecting the source with the stcrt instruction. Each core has its own 'rxtx' register that only it can write.

It is possible to select between two different ways of receiving the data. Table 13 "ch_rxtx internal register in read mode for source sssc to ospc" shows the register in read mode when the data from one single microcore is selected. This allows transmitting 16-bit data between one microcore and another.

Table 14 "ch_rxtx internal register in read mode for source sumh, suml" shows the register in read mode when the source "sumh" or "suml" is selected. In this mode, the bits H0 to H3 or L0 to L3 in all four microcores ch_rxtx registers are counted and the result can be read from the communication register. The result for each bit Hx and Lx can be between 0 ("0000") and 6 ("0110").

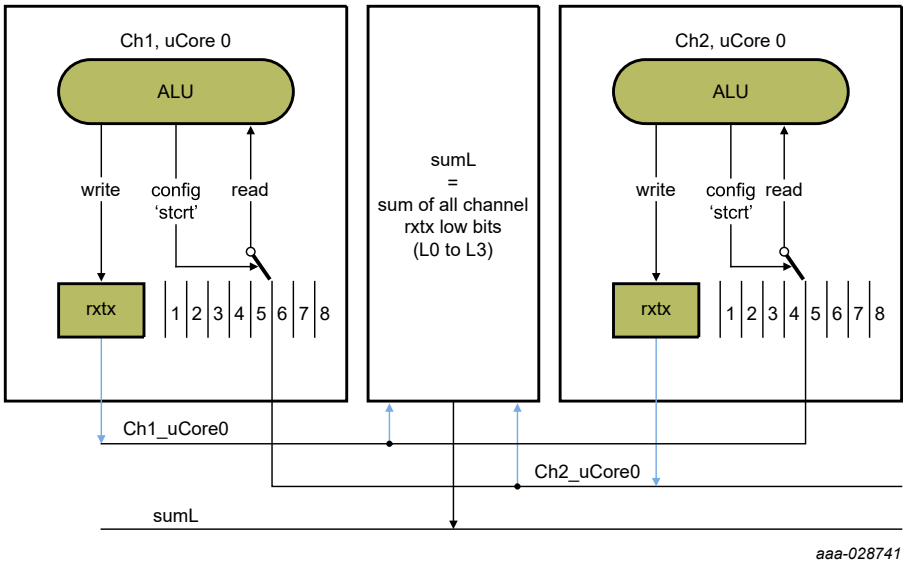

Figure 4. Communication example between Ch1 uc0 and Ch2 uc0 (sspc)

Table 12. ch_rxtx internal register in write mode

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Transmit_data															
Bits in sumh or suml mode	H3	H2	H1	H0	L3	L2	L1	L0	—							
Reset	0000 0000 0000 0000															

Table 13. ch_rxtx internal register in read mode for source sssc to ospc

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Received_data_from_selected_microcore															
R/W	R/W															
Lock	yes															
Reset	0000 0000 0000 0000															

Table 14. ch_rxtx internal register in read mode for source sumh, suml

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	sum(H3)or sum(L3)				sum(H2)or sum(L2)				sum(H1) or sum(L1)				sum(H0) or sum(L0)			
Reset	0000 0000 0000 0000															

Table 15. Inter-communication instruction set

stcrt	Set channel communication register

2.13 Shortcuts

Shortcuts are used to connect a core to the hardware. There are two types of shortcuts; 'output driver' shortcuts (dfcsct) and 'current sense block' shortcuts (dfsct). Output driver shortcuts allow a core to modify the states of up to three outputs at once. By modifying all three outputs in a single instruction, fully synchronized driver changes can occur in a single instruction.

Each core has one current sense block shortcut. The current sense block shortcut connects the core to one of the six current sense blocks. This shortcut is used primarily for testing the 'own current' current threshold (see the 'ocur' field value of the 'jocf' and 'jocr' instructions) or waiting for the 'own current' threshold to be reached (see the 'wait' instruction's 'ocur' field value.)

Another benefit of shortcuts is the ability to write core-independent code. This allows the exact same code to operate on different sets of output drivers and current sense blocks without having to make driver specific conditional jumps.

Table 16. Shortcuts definition instructions

dfcsct	Define current shortcut
dfsct	Define predriver output shortcuts

2.14 Current sense blocks

As described in the data sheet, current sense blocks can be used as a current sense with offset compensation, or as an ADC or in a DCDC mode. The following instructions in this section are used to configure each mode.

It is possible to set the DCDC mode (stdcctl) from the microcode of any core, as long as the core has access to the LS7 output. The low-side used to control the DCDC low-side (LS7) has to be defined as shortcut 2 (dfsct) in order to use the DCDC mode.

The DAC registers are used to setup the current measurement block DACs. These DACs are affected as shown in Table 17 "Current measurement DACs affectation to

microcores". These DACs can be set by using the stdm instruction to setup the access mode. The DAC registers can be loaded with the cp and load instructions. Loading DAC register values to other registers or DRAM is also possible by means of the cp and store instructions.

Table 17. Current measurement DACs affectation to microcores

Microcore	dacsssc	dacossc	*oc_dac_sel_ucX* = '0' (next channel)		*oc_dac_sel_ucX* = '1' (previous channel)	
			dacssoc	dacosoc	dacssoc	dac osoc
Uc0, channel 1	dac1	dac2	dac3	dac4	dac5	dac6
Uc1, channel 1	dac2	dac1	dac4	dac3	dac6	dac5
Uc0, channel 2	dac3	dac4	dac5	dac6	dac1	dac2
Uc1, channel 2	dac4	dac3	dac6	dac5	dac2	dac1

Table 18. Current sense instructions

stadc	Set ADC mode
stdcctl	Set DC-DC control mode
stdm	Set DAC register mode access
stdm	Set current measure operational amplifier gain
stoc	Set offset compensation

2.15 Output drivers

The instructions described in this section are used to control the output drivers. Each high-side and low-side can be turned ON/OFF by all microcores if the output access registers are configured properly (refer to register 184h to 187h). Low-side and high-side bias needs to be enabled before using the diagnostics, and can be kept ON during the application.

It is possible to enable or disable the end-of-actuation mode. In the final phase of an actuation, while the current in the actuator is decreasing, it is possible to detect when the current has reached the zero value. In most applications, it is required that the Vsource threshold for the corresponding HS output is set to zero. This condition can be automatically enabled and disabled together with the end-of-actuation mode.

Table 19. Output drivers instructions

bias	Enable high-side and low-side bias
steoa	Set end-of-actuation mode
stfw	Set freewheeling mode
sto	Set single predriver output
stos	Set predriver output shortcuts
stslew	Set predriver output slew rate mode
Section 3.2.2	Set measurement function multiplexers (only available on PT2001M)

2.16 Interrupts

An interrupt routine is executed when an interrupt request is received by the microcore. The microcore must not have already been executing another interrupt routine. The interrupt routine cannot be interrupted by any other interrupt, but only be terminated via an iret instruction or (if configured in this way by the iconf instruction) by reading the related diagnosis register through SPI (not through the SPI back door):

- Err_ucXchY registers (162h to 169h) for the automatic diagnosis interrupt
- Driver_status register (1D2h) for the disabled drivers interrupt.

The interrupts received are queued while another interrupt execution is ongoing. When exiting the ongoing interrupt routine with the iret instruction, the queue can be cleared and queued interrupts are ignored. Otherwise, the queued interrupts are executed according to their priorities:

- Automatic diagnosis interrupt (higher priority)
- Driver disabled interrupt
- Software interrupt (lower priority)

The interrupt return address is always calculated when the interrupt occurs, and is stored in the Ucx_irq_status registers (10Fh, 110h, 12Fh, 130h). The return address is the address where the code execution was interrupted. If a wait or a conditional jump instruction is interrupted, the return address is defined, restoring the status of the feedback at the moment the interrupt request occurred.

2.16.1 Automatic interrupt

Automatic diagnosis interrupt routine address: this address (defined in the Diag_routine_addr (10Ch and 12Ch) section) is selected as the new uPC value if an automatic diagnosis interrupt request is received by the microcore. This condition has an higher priority than any instruction and any other interrupt.

The following instructions are used to enable or disable the automatic diagnostics and select different configuration.

Before turning ON the diagnostics, the error table needs to be configured properly by the SPI (refer to registers HSx_output_config (153h to 161h) and LSx_output_config (140h to 152h)). The threshold can be configured either by the SPI (refer to registers V_{DS} and V_{SRC} threshold section in the data sheet) or by using the chth instruction.

Table 20. Automatic diagnostics instructions

chth	Change V_{DS} and V_{SRC} threshold
endiag	Enable automatic diagnosis
endiaga	Enable all automatic diagnosis
endiags	Enable automatic diagnosis shortcuts
slfbk	Select HS2/4/6 feedback reference

2.16.2 Driver disable interrupt

Driver disabled interrupt routine address: this address (defined in the Driver_disabled_routine_addr (10Dh and 12Dh) section) is selected as the new uPC value if an interrupt request, due to disabled drivers, is received by the microcore. This condition has an higher priority than any instruction and the software interrupt.

2.16.3 Software interrupt

Software interrupt routine address: this address (defined in the Sw_interrupt_routine_addr (10Eh and 12Eh) section) is selected as the new uPC value if a software interrupt request is received by the microcore. This condition has a higher priority than any instruction.

Table 21. Software interrupt Instructions

reqi	Software interrupt request
swi	Enable/disable software interrupt
iret	Return from interrupt

2.17 Counter/timers

This block contains 4 pairs of 16-bit up counter and 16-bit end-of-count registers. Each of the four counters is compared with an eoc_reg (end-of-count register); if the counter is greater or equal than its corresponding end-of-count, then a terminal count signal is asserted. These signals are fed to uinstruction_rom.

At reset each counter and eoc_reg is set to zero. When a counter reaches its end-of-count value, its value does not increase. If the eoc_reg is changed without resetting the counter value, the counter value starts to increase again (if the new end-of-count value is greater than the counter value) until the new end-of-count value is reached.

These counters can be loaded with data coming from the DRAM or from the internal bus (e.g. ALU registers). Also, the counters can write data into the DRAM or into any of the registers connected to the internal bus (this function can be used to perform period measurements on the input signals).

It is possible to update any terminal count register without stopping the associated counter. This feature allows on-the-fly data correction in the actuated timings. All load instructions executed can simultaneously load the eoc_reg with the value specified in the microinstruction and reset the counter.

The counter starts counting up until it meets the eoc_reg value: at this point an eoc (end-of-count) signal is set to inform the microprogram that this event has occurred. There are also load instructions that do not reset the counter after loading the eoc_reg register. See Section 3.2 "Instruction set" for details of all the instructions.

Counter 1 and 2 always operate with the ck execution clock: so the maximum time that is possible to measure with a single counter is 2^{16} * ck clock period (10,923 ms at 6.0 MHz). Counter 3 and 4 can operate with a slower clock, obtained dividing the execution clock frequency (by an integer factor from 1 to 12, 14, 16, 32, or 64), to measure longer times with lower resolution (refer to register Counter_34_prescaler (111h and 131h).

Table 22. Load counter and set output instructions

ldca	Load counter from ALU register and set outputs
ldcd	Load counter from Data RAM and set outputs

2.18 SPI back door

All the SPI accessible registers can be accessed also by the microcores through an "SPI back door". Note that both Data and Code RAMs are unavailable through the back

door. The spi_access_controller receives all the register read/write requests, from the SPI interface and from all the enabled microcores. Top priority is given to the requests coming from the SPI interface.

To read a SPI register, first the eight LSBs of the address must be provided in the eight LSBs of the 'SPI address' at an internal memory map address to the load instruction. A read operation must be requested with the rdspi instruction. The result is available at the 'SPI data' address of the internal memory map. Note that it is necessary to wait one clk cycle to make sure the spi_data register is updated.

To write to a SPI register, first the eight LSBs of the address must be provided in the eight LSBs of the 'SPI address' address, and the data to write must be provided at the 'SPI data' address to the load instruction. A write operation must be requested with the wrspi instruction. Both the SPI read and write operations are two cycle operations. The registers must not be changed while the operation is in progress.

If the SPI back door is not used, the 8-bit register at the address 'SPI address' and the 16-bit register at the address 'SPI data' can be used as spare register.

Table 23. SPI back door instructions

rdspi	SPI read request
slsa	Select SPI address
wrspi	SPI write request

3 Instruction set and subsets

3.1 Internal registers operand subsets

This section details the predefined microcore register subsets used as instruction operands in Direct Addressing mode (DM).

Table 24. Operand subset overview

Operand label	Operand subset description
AluReg	Register designator for registers r0, r1, r2, r3, r5, r5, ir, mh, and ml
AluGprIrReg	Register designator for registers r0, r1, r2, r3, r5, r5, and ir
UcReg	Register designator for registers r0, r1, r2, r3, r5, r5, ir, mh, ml, ar (arith_reg), aux, jr1, jr2, cnt1, cnt2, cnt3, cnt4, eoc1, eoc1, eoc3, eoc4, flag, cr (ctrl_reg), sr (status_bits), spi_data, dac_sssc, dac_ossc, dac_ssoc, dac_osoc/batt, dac4h4n/boost, spi_add, irq (irq_status), and rxtx (ch_rxtx)
JpReg	Register designator for registers jr0 and jr1

3.1.1 AluReg subset

Table 25. AluReg subset description

Register label	Operand binary value
r0	000
r1	001
r2	010
r3	011

Register label	Operand binary value
r4	100
ir	101
mh	110
ml	111

3.1.2 AluGprIrReg subset

Table 26. AluGpsIrReg subset description

Register label	Operand binary value
r0	000
r1	001
r2	010
r3	011
r4	100
ir	101

3.1.3 UcReg subset

Table 27. UcReg subset description

Register label	Operand binary value
r0	00000
r1	00001
r2	00010
r3	00011
r4	00100
ir	00101
mh	00110
ml	00111
ar[1]	01000
aux	01001
jr1	01010
jr2	01011
cnt1	01100
cnt2	01101
cnt3	01110
cnt4	01111
eoc1	10000
eoc2	10001
eoc3	10010

Register label	Operand binary value
eoc4	10011
flag	10100
cr[2]	10101
sr[3]	10110
spi_data	10111
dac_sssc	11000
dac_ossc	11001
dac_ssoc	11010
dac_osoc	11011
dac4h4n	11100
spi_add	11101
irq[4]	11110
rxtx [5]	11111

[1] ar is the ALU arithmetic register arith_reg
[2] cr is the control register ctrl_reg
[3] sr is the status bits register status_bits
[4] irq is the interrupt status register irq_status
[5] rxtx is the other channel communication register ch_rxtx

3.1.4 JpReg subset

Table 28. JrReg subset description

Register label	Operand binary value
jr1	0
jr2	1

3.2 Instruction set

The instructions contain an entry for each assembler mnemonic, in alphabetic order. Figure 5 is a representation of an instruction page.

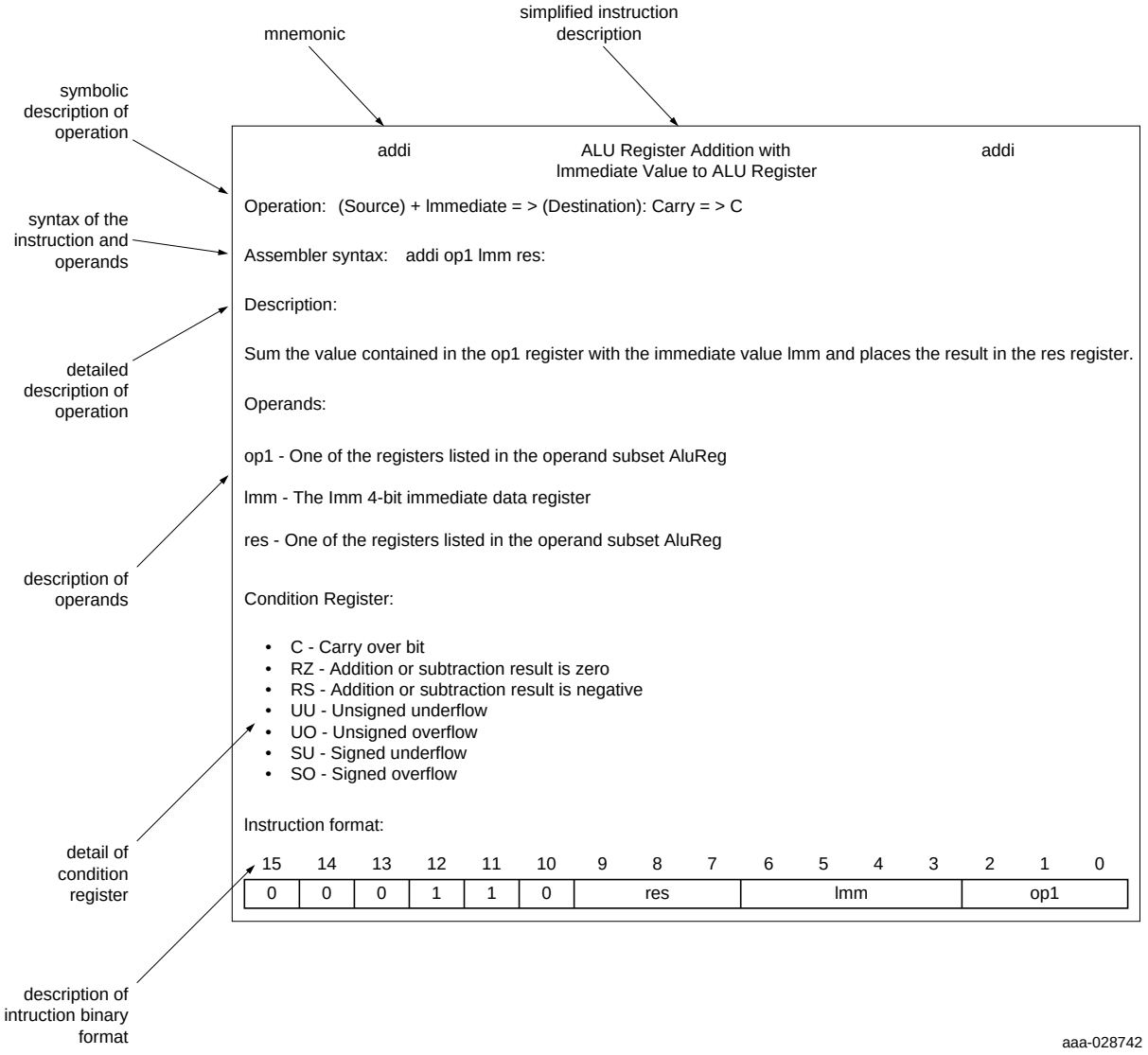

Figure 5. Description of instruction page

3.2.1 Mnemonic index

This subsection contains an entry for each assembler mnemonic, in alphabetic order.

Table 29. Instruction index

Instruction name	Instruction description
add	Add two ALU registers and place the result in one of the ALU registers
addi	Add an ALU register to the value in the immediate register and place the result in an ALU register
and	AND-mask on ALU register with the immediate register to ALU register
bias	Enable high-side and low-side bias
chth	Change V_{DS} and V_{SRC} threshold
cp	Copy source register data in destination register

Instruction name	Instruction description
cwef	Create wait table entry far
cwer	Create wait table entry relative
dfcsct	Define current shortcut
dfsct	Define predriver output shortcuts
endiag	Enable automatic diagnosis
endiaga	Enable all automatic diagnosis
endiags	Enable automatic diagnosis shortcuts
iconf	Interrupt configuration
iret	Return from interrupt
jarf	Jump far on arithmetic condition
jarr	Jump relative on arithmetic condition
jcrf	Jump far on control register condition
jcrr	Jump relative on control register condition
jfbkf	Jump far on feedback condition
jfbkr	Jump relative on feedback condition
jmpf	Unconditional jump far
jmpr	Unconditional jump relative
jocf	Jump far on condition
jocr	Jump relative on condition
joidf	Jump far on microcore condition
joidr	Jump relative on microcore condition
joslf	Jump far on start condition
joslr	Jump relative on start condition
jsrf	Jump far on status register bit condition
jsrr	Jump relative on status register bit condition
jtsf	Jump far to subroutine
jtsr	Jump relative to subroutine
ldca	Load counter from ALU register and set outputs
ldcd	Load counter from Data RAM and set outputs
ldirh	Load 8-MSB ir register
ldirl	Load 8-LSB ir register
ldjr1	Load jump register 1
ldjr2	Load jump register 2
load	Load data from Data RAM to register
mul	Multiply two ALU registers and place the result in reg32
muli	Multiply an ALU register with the value in the immediate register and place the result in reg32

Instruction name	Instruction description
not	Invert ALU register bits
or	OR mask on ALU register with immediate register to ALU register
rdspi	SPI read request
reqi	Software interrupt request
rfs	Return from subroutine
rstreg	Reset registers (control, status, automatic diagnostics...)
rstsl	Start-latch registers reset
sh32l	Shift left multiplication result register
sh32li	Shift left multiplication result register of immediate value
sh32r	Shift right multiplication result register
sh32ri	Shift right multiplication result register of immediate value
shl	Shift left ALU register
shl8	Shift left ALU register of 8 bits
shli	Shift left the ALU register of immediate value
shls	Shift left signed ALU register
shlsi	Shift left signed ALU register of immediate value
shr	Shift right ALU register
shr8	Shift right ALU register of 8 bits
shri	Shift right the ALU register of immediate value
shrs	Shift right signed ALU register
shrsi	Shift right signed ALU register of immediate value
slab	Select Data RAM address base
slfbk	Select HS2/4/6 feedback reference
slfbks	Select feedback for V_{DS} HS1 to HS5 based on shortcut
slsa	Select SPI address
stab	Set Data RAM address base
stadc	Set ADC mode
stal	Set arithmetic logic mode
stcrb	Set control register bit
stcrt	Set channel communication register
stdcctl	Set DC-DC control mode
stdm	Set DAC register mode access
stdrm	Set Data RAM read mode
steoa	Set end-of-actuation mode
stf	Set flag
stfw	Set freewheeling mode
stgn	Set current measure operational amplifier gain

Instruction name	Instruction description
stirq	Set IRQB pin
sto	Set single predriver output
stoc	Set offset compensation
store	Store register data in Data RAM
stos	Set predriver output shortcuts
stslew	Set predriver output slew rate mode
stmfm	Set measurement function MUX
stsrb	Set status register bit
sub	Subtract two ALU registers and place the result in one of the ALU registers
subi	Subtract the value in the immediate register from an ALU register and place the result in an ALU register
swap	Swap bytes inside ALU register
swi	Enable / disable software interrupt
toc2	Convert an integer in an ALU register to 2's compliment format
toint	Convert the two's complement value contained in an ALU register to integer format
wait	Wait until condition satisfied
wrspi	SPI write request
xor	Mask XOR with immediate register

add

Description:

Sums the value contained in the op1 register with the value contained in op2 register and places the result in the res register.

Operation: (Source1) + (Source2) → (Destination); Carry → C

Assembler syntax: `add op1 op2 res;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- res – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- C – Carry over bit
- RZ – Addition or subtraction result is zero
- RS – Addition or subtraction result is negative
- UU – Unsigned underflow
- UO – Unsigned overflow
- SU – Signed underflow
- SO – Signed overflow

Table 30. add instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	res			0	op2			op1		

addi

Description:

Sums the value contained in the op1 register with the immediate value Imm and places the result in the res register.

Operation: (Source) + Immediate value → (Destination); Carry → C

Assembler syntax: `addi op1 Imm res;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- res – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Table 31. addi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	res			Imm				op1		

and

Description:

Applies the AND-mask contained in the Ir register to the value contained in the op1 register. The result is placed in the op1 register. The initial data stored in the op1 register is lost.

Operation: (Source) + Immediate register → (Source)

Assembler syntax: `and op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Ir – The ALU immediate register

Condition register:

- MN – Mask result is 0000h
- MM – Mask result is FFFFh

Table 32. and instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	1	0	0	1	op1		

Source code example:

```
*### Do 0x0C00 & IRQ status register ####
irq_and:    cp irq r0;          * Save irq register into r0 (for this example irq = 0x400 due to a sw interrupt 1)
            ldirh 0Ch rst;      * Load immediate register ir MSB with 0x0C and reset the LSB -> IR = 0x0C00
            and r0;             * Operation does ir & r0 = 0x0C00 & 0x0400 = 0x0400 and save this results in r0
```

bias

Description:

Enables/disables individually the high-side and low-side PT2001 load bias structures. This operation is successful only if the microcore has the right to drive the output related to the selected bias structure. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

Assembler syntax: `bias BiasTarget Ctrl;`

Operands:

BiasTarget– Operand defines the bias structure(s) to be selected

Operand label	Operand description
hs1	Select HS1 bias structure
hs2	Select HS2 bias structure
hs3	Select HS3 bias structure
hs4	Select HS4 bias structure
hs5	Select HS5 bias structure
ls1	Select LS1 bias structure
ls2	Select LS2 bias structure
ls3	Select LS3 bias structure
ls4	Select LS4 bias structure
ls5	Select LS5 bias structure
ls6	Select LS6 bias structure
hs2s	Select HS2 strong bias structure
hs4s	Select HS4 strong bias structure
all	Select all high-side and low-side predriver bias structures including strong bias structures
hs	Select all high-side predriver bias structures including strong bias structures
ls	Select all low-side predriver bias structures

Operands:

- Ctrl – Operand defines the bias structure(s) state to be applied

Operand label	Operand description
off	Bias structure disable
on	Bias structure enable

Table 33. bias instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	0	1	Ctrl	BiasTarget			

chth

Description:

Changes the thresholds for the selected V_{DS} and V_{SRC} feedback comparator.

These are the same values as in registers Vds_threshold_hs_partx (16Bh, 16Ch), Vsrc_threshold_hs_partx (16Dh, 16Eh) and Vds_threshold_ls_partx (16Fh, 170h).

This operation is successful only if the microcore has the right to drive the output related to selected threshold. The configuration of the high-side predriver V_{SRC} threshold is also impacted by the bootstrap initialization mode.

Assembler syntax: `chth SelFbk ThLevel;`

Operands:

- SelFbk – Operand defines the threshold comparator to be selected
- ThLevel – This operand specifies one of the 16 threshold level values

Operand label	Operand description
SelFbk operand	
hs1v	HS1 V_{DS} feedback
hs1s	HS1 V_{SRC} feedback
hs2v	HS2 V_{DS} feedback
hs2s	HS2 V_{SRC} feedback
hs3v	HS3 V_{DS} feedback
hs3s	HS3 V_{SRC} feedback
hs4v	HS4 V_{DS} feedback
hs4s	HS4 V_{SRC} feedback
hs5v	HS5 V_{DS} feedback
hs5s	HS5 V_{SRC} feedback
ls1v	LS1 V_{DS} feedback
ls2v	LS2 V_{DS} feedback
ls3v	LS3 V_{DS} feedback
ls4v	LS4 V_{DS} feedback
ls5v	LS5 V_{DS} feedback
ls6v	LS6 V_{DS} feedback
ThLevel operand	
lv1	First level: 0.00 V
lv2	Second level: 0.50 V
lv3	Third level: 1.0 V
lv4	Fourth level: 1.5 V
lv5	Fifth level: 2.0 V
lv6	Sixth level: 2.5 V
lv7	Seventh level 3.0 V

Operand label	Operand description
lv8	Eighth level: 3.5 V
lv9	Ninth level: 0.10 V
lv10	Tenth level: 0.20 V
lv11	Eleventh level: 0.30 V
lv12	Twelfth level: 0.40 V
lv13	Thirteenth level: 0.60 V
lv14	Fourteenth level: 0.70 V
lv15	Fifteenth level: 0.80 V
lv16	Sixteenth level: 0.90 V

Table 34. chth instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	1	1	SelFbk				ThLevel		

cp

Description:

Copies the value from the source register op1 into the destination register op2.

Assembler syntax: `cp op1 op2;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.3 "UcReg subset"
- op2 – One of the registers listed in the operand Section 3.1.3 "UcReg subset"

Table 35. cp instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	op1					op2					0	0	0

cwef

Description:

Initializes or changes a row in the wait table used by the wait instruction The wait table is a five-row/two-column table:

- The first column contains the wait conditions.
- The second column contains the jump register name op1 contains the absolute destination addresses. Up to five conditions may be checked at the same time.

When the condition Cond is satisfied and the entry is enabled, the execution continues either to the address 'jr1' or 'jr2' as specified by the op1 parameter.

Assembler syntax: `cwef op1 Cond Entry;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- Cond – Operand defines the condition to be satisfied to enable the jump far

Operand label	Operand description
Cond operand	
_f0	Flag 0 low
_f1	Flag 1 low
_f2	Flag 2 low
_f3	Flag 3 low
_f4	Flag 4 low
_f5	Flag 5 low
_f6	Flag 6 low
_f7	Flag 7 low
_f8	Flag 8 low
_f9	Flag 9 low
_f10	Flag 10 low
_f11	Flag 11 low
_f12	Flag 12 low
_f13	Flag 13 low
_f14	Flag 14 low
_f15	Flag 15 low
f0	Flag 0 high
f1	Flag 1 high
f2	Flag 2 high
f3	Flag 3 high
f4	Flag 4 high
f5	Flag 5 high
f6	Flag 6 high

Operand label	Operand description
f7	Flag 7 high
f8	Flag 8 high
f9	Flag 9 high
f10	Flag 10 high
f11	Flag 11 high
f12 / cur4	Flag 12 high / current feedback high
f13	Flag 13 high
f14	Flag 14 high
f15	Flag 15 high
tc1	Terminal count 1
tc2	Terminal count 2
tc3	Terminal count 3
tc4	Terminal count 4
_start	Start low
start	Start high
_sc1v	Shortcut1 V_{DS} feedback low
_sc2v	Shortcut2 V_{DS} feedback low
_sc3v	Shortcut3 V_{DS} feedback low
_sc1s	Shortcut1 source feedback low
_sc2s	Shortcut2 source feedback low
_sc3s	Shortcut3 source feedback low
sc1v	Shortcut1 V_{DS} feedback high
sc2v	Shortcut2 V_{DS} feedback high
sc3v	Shortcut3 V_{DS} feedback high
opd	Instruction request to ALU executed
vb	Boost voltage high
_vb	Boost voltage low
cur1	Current feedback 1 high
cur2	Current feedback 2 high
cur3	Current feedback 3 high
cur4l	Current feedback 4l high
cur4h	Current feedback 4h high
cur4n	Current feedback 4n high
_cur1	Current feedback 1 low
_cur2	Current feedback 2 low
_cur3	Current feedback 3 low
_cur4l	Current feedback 4l low

Operand label	Operand description
_cur4h	Current feedback 4h low
_cur4n	Current feedback 4n low
ocur	Own current feedback high
_ocur	Own current feedback low
Entry operand	
row1	Wait table row 1
row2	Wait table row 2
row3	Wait table row 3
row4	Wait table row 4
row5	Wait table row 5
row6	Wait table row 6

Table 36. cwef instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	op1	Entry			Cond					

```
InitEntry:      cwer vboost_high_hit vb row1;     * Set wait table entry 1in case Vboost voltage is higher than Vboost_dac
                cwer current_high_hit ocur row2;  * Set wait table entry 2 in case own current sense is higher than dac threshold
                ldjr1 eoinj0;                     * Set jr1 register to jump to eoinj0 label because there are more than 15
                                                    instructions between this instruction and the label eoinj0
                cwef jr1 _start row3              * Set wait table entry 3 in case start pin is going low
                wait row123                       * Wait here until one of the three conditions is satisfied
vboost_high_hit:
* #### Add some code here #####
current_high_hit:
*#### Add some code here #####
eoinj0:                                           *More than 15 lines between the wait declaration and this label
*#### Add some code here #####
```

cwer

Description:

Initializes or changes a row in the wait table used by the wait instruction The wait table is a six-row/two-column table:

- The first column contains the wait conditions
- The second column contains the destination jump addresses

Up to five conditions can be checked at the same time.

When the condition Cond is satisfied and the entry is enabled, the execution continues at the correspondent destination jump address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `cwer Dest Cond Entry ;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}
- Cond – Operand defines the condition to be satisfied to enable the jump far

Operand label	Operand description
Cond operand	
_f0	Flag 0 low
_f1	Flag 1 low
_f2	Flag 2 low
_f3	Flag 3 low
_f4	Flag 4 low
_f5	Flag 5 low
_f6	Flag 6 low
_f7	Flag 7 low
_f8	Flag 8 low
_f9	Flag 9 low
_f10	Flag 10 low
_f11	Flag 11 low
_f12	Flag 12 low
_f13	Flag 13 low
_f14	Flag 14 low
_f15	Flag 15 low
f0	Flag 0 high
f1	Flag 1 high
f2	Flag 2 high

Operand label	Operand description
f3	Flag 3 high
f4	Flag 4 high
f5	Flag 5 high
f6	Flag 6 high
f7	Flag 7 high
f8	Flag 8 high
f9	Flag9 high
f10	Flag 10 high
f11	Flag 11 high
f12	Flag 12 high / current feedback high
f13	Flag 13 high
f14	Flag 14 high
f15	Flag 15 high
tc1	Terminal count 1
tc2	Terminal count 2
tc3	Terminal count 3
tc4	Terminal count 4
_start	Start low
start	Start high
_sc1v	Shortcut1 V_{DS} feedback low
_sc2v	Shortcut2 V_{DS} feedback low
_sc3v	Shortcut3 V_{DS} feedback low
_sc1s	Shortcut1 source feedback low
_sc2s	Shortcut2 source feedback low
_sc3s	Shortcut3 source feedback low
sc1v	Shortcut1 V_{DS} feedback high
sc2v	Shortcut2 V_{DS} feedback high
sc3v	Shortcut3 V_{DS} feedback high
opd	Instruction request to ALU executed
vb	Boost voltage high
_vb	Boost voltage low
cur1	Current feedback 1 high
cur2	Current feedback 2 high
cur3	Current feedback 3 high
cur4l	Current feedback 56l high
cur4h	Current feedback 56h high
cur4n	Current feedback 4n high

Operand label	Operand description
_cur1	Current feedback 1 low
_cur2	Current feedback 2 low
_cur3	Current feedback 3 low
_cur4l	Current feedback 56l low
_cur4h	Current feedback 56h low
_cur4n	Current feedback 4n low
ocur	Own current feedback high
_ocur	Own current feedback low
Entry operand	
row1	Wait table row 1
row2	Wait table row 2
row3	Wait table row 3
row4	Wait table row 4
row5	Wait table row 5
row6	Wait table row 6

Table 37. instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	Dest					Entry			Cond					

Source code example:

```
InitEntry:      cwer vboost_high_hit vb row1;     * Set the wait table entry 1in case Vboost voltage is higher than Vboost_dac
                cwer current_high_hit ocur row2;  * Set the wait table entry 2 in case own current sense is higher than dac threshold
                ldjr1 eoinj0;                     * Set jr1 register to jump to eoinj0 label because there is more than 15 instructions between this  q
                                                  instruction and the label eoinj0
                cwef jr1 _start row3              * Set the wait table entry 3 in case start pin is going low
                wait row123                       * Wait here until one of the three condition is reached
vboost_high_hit:
* #### Add some code here #####
current_high_hit:
*#### Add some code here #####
eoinj0:                                           *More than 15 lines between the wait declaration and this label
*#### Add some code here #####
```

dfcsct

Description:

Defines the shortcut for the current feedback.

This shortcut defines the connection between the physical current feedback input of the microcore and the current measurement block. At reset, the default shortcut setting is the following:

Shortcut	Uc0Ch1	Uc1Ch1	Uc0Ch2	Uc1Ch2
ShrtCur	dac1	dac2	dac3	dac4

Assembler syntax: `dfcsct ShrtCur;`

Operands:

- ShrtCur – Operand defines which current measurement block is dedicated to the shortcut

Operand label	Operand description	Operand binary value
dac1	DAC1 is selected as current shortcut	00
dac2	DAC2 is selected as current shortcut	01
dac3	DAC3 is selected as current shortcut	10
dac4l	DAC4 is selected as current shortcut	11

Table 38. dfcsct instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	1	1	1	0	ShrtCur	

dfsct

Description:

Defines three shortcuts applied to three predriver outputs among the set of all the low-side and high-side predrivers.

The shortcuts table defines the connection between the physical outputs of the microcore and the external outputs pin (G_HSx and G_LSx) driving the MOSFETs.

Note that in order to use the async or sync mode, the low-side use for DCDC has to be set as shortcut 2 (ex: dfsct undef ls7 undef).

At reset the default shortcut setting is undefined

Assembler syntax: `dfsct Shrt1 Shrt2 Shrt3;`

Operands:

- Shrt1, Shrt2, and Shrt3 – Operands define to which predriver the shortcut is dedicated

Operand label	Operand description
hs1	High-side predriver 1
hs2	High-side predriver 2
hs3	High-side predriver 3
hs4	High-side predriver 4
hs5	High-side predriver 5
ls1	Low-side predriver 1
ls2	Low-side predriver 2
ls3	Low-side predriver 3
ls4	Low-side predriver 4
ls5	Low-side predriver 5
ls6	Low-side predriver 6
ls7	Low-side predriver 7
undef	Undefined shortcut

Table 39. dfsct instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	Shrt1				Shrt2				Shrt3				1	1

endiag

Description:

Enables or disables the automatic diagnosis for a single output and the related interrupt procedure for error handling.

This operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

At reset the automatic diagnosis is disabled.

Assembler syntax: `endiag Sel Diag;`

Operands:

- Sel – Operand defines the monitored predriver and V_{DS} or V_{SRC} feedback

Operand label	Operand description
hs1v	High-side predriver 1 V_{DS} feedback
hs1s	High-side predriver 1 S_{RC} feedback
hs2v	High-side predriver 2 V_{DS} feedback
hs2s	High-side predriver 2 S_{RC} feedback
hs3v	High-side predriver 3 V_{DS} feedback
hs3s	High-side predriver 3 S_{RC} feedback
hs4v	High-side predriver 4 V_{DS} feedback
hs4s	High-side predriver 4 S_{RC} feedback
hs5v	High-side predriver 5 V_{DS} feedback
hs5s	High-side predriver 5 S_{RC} feedback
ls1v	Low-side predriver 1 V_{DS} feedback
ls2v	Low-side predriver 2 V_{DS} feedback
ls3v	Low-side predriver 3 V_{DS} feedback
ls4v	Low-side predriver 4 V_{DS} feedback
ls5v	Low-side predriver 5 V_{DS} feedback
ls6v	Low-side predriver 6 V_{DS} feedback
diagoff	Automatic diagnosis disable
diagon	Automatic diagnosis enable

Table 40. endiag instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	1	1	0	Sel				Diag

endiaga

Description:

Enables or disables the automatic diagnosis for all the predrivers output the microcore is configured to drive. If automatic diagnosis condition is satisfied, the related interrupt procedure for error handling is triggered.

The operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

At reset the automatic diagnosis is disabled.

Assembler syntax: `endiaga Diag;`

Operands:

- Diag – Operand defines the diagnosis status

Operand label	Operand description	Operand binary value
diagoff	Automatic diagnosis disable	0
diagon	Automatic diagnosis enable	1

Table 41. endiaga instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	Diag

endiags

Description:

Enables or disables the automatic for the outputs selected via shortcuts. Four events can be monitored in parallel:

- The drain-source voltage on shortcut1 output (Diag_sh1_vds)
- The source voltage on shortcut1 output (Diag_sh1_src)
- The drain-source voltage on shortcut2 output (Diag_sh2_vds)
- The drain-source voltage on shortcut3 output (Diag_sh3_vds)

If automatic diagnosis condition is satisfied, the related interrupt procedure for error handling is triggered. The shortcuts are defined with the dfsct instruction.

The operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

At reset the automatic diagnosis are disabled.

Assembler syntax: `endiags Diag_sh1_vds Diag_sh1_src Diag_sh2_vds Diag_sh3_vds;`

Operands:

- Diag _sh1_vds, Diag_sh2_vds and Diag_sh3_vds – Operands corresponding to the shortcuts related to V_{DS} to be monitored.

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Automatic diagnosis disabled
on	Automatic diagnosis enabled

Diag_sh1_src – Operand corresponding to the shortcuts related to V_{SRC} to be monitored.

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Automatic diagnosis disabled
on	Automatic diagnosis enabled

Table 42. endiags instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	0	Diag_ sh1_vds		Diag_sh1_src		Diag_ sh2_vds		Diag_ sh3_vds	

iconf

Description:

Configures the microcore to be enabled by the interrupt return request.

The automatic interrupt return request is issued from, according to the iret_en bit state of the Driver_config register (1C5h):

- Re-enabling the drivers in case the disabled drivers interrupt
- Reading or writing the Driver_status register (1D2h) in case of automatic diagnosis interrupt. This register must be configured such as to be 'reset at read'.

The reset value is none.

Assembler syntax: `iconf Conf;`

Operands:

- Conf – Operand defines interrupt behaviors

Operand label	Operand description
none	The microcore ignores all automatic interrupt return request
continue	When an interrupt return request is received, the code execution continues from where it was interrupted
restart	When an interrupt return request is received, the code execution restarts from the entry point

Table 43. iconf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	0	0	0	0	1	Conf	

iret

Description:

Ends the interrupt routine and clears the microcore interrupt register (uc0_irq_status (10Fh and 12Fh) and uc1_irq_status (110h and 130h)).

Assembler syntax: `iret Type Rst;`

Operands:

- Type – Operand defines how the program counter (uPC) is handled returning from the interrupt routine

Operand label	Operand description
continue	The execution is resumed at the address stored in the 10 LSBs of the interrupt register
restart	The execution is resumed at the address stored in the uc0_entry_point (10Ah and 12Ah) or uc1_entry_point (10Bh and 12Bh)

Rst – Operand defines if the pending interrupt queue is clear when the iret instruction is executed

Operand label	Operand description
_rst	The pending interrupts queue is not cleared
rst	The pending interrupts queue is cleared

Table 44. iret instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	1	0	Type	Rst

jarf

Description:

Configures the jump to absolute location on arithmetic condition.

If the condition defined by the BitSel operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `jarf op1 BitSel;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- BitSel – Operand defines the arithmetic condition that triggers the jump. The arithmetic conditions are stored into the ALU condition register.

Operand label	Operand description	Operand binary value
opd	OD – Operation complete	0000
ovs	SO – Overflow with signed operands	0001
uns	SU – Underflow with signed operands	0010
ovu	UO – Overflow with unsigned operands	0011
unu	UU – Underflow with unsigned operands	0100
sgn	CS – Sign of result	0101
zero	RZ – Result is zero	0110
mloss	ML – Multiply precision loss	0111
mover	MO – Multiply overflow	1000
all1	MM – Result of mask operation is FFFFh	1001
all0	MN – Result of mask operation is 0000h	1010
aritl	false if add/sub saturation is enabled, true otherwise (see stal instruction)	1011
arith	false if logic is set to two's complement, true if logic is set to positive numbers only (see stal instruction)	1100
carry	C – Carry	1101
conv	CS – Conversion sign	1110
csh	SB – Carry on shift operation	1111

Table 45. jarf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	BitSel				op1	0	1	0	1

jarr

Description:

Configures jump to relative location on arithmetic condition.

If the condition defined by the BitSel operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `jarr Dest BitSel;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}.
- BitSel – Operand defines the arithmetic condition that triggers the jump. The arithmetic conditions are stored into the ALU condition register.

Operand label	Operand description	Operand binary value
opd	OD -Operation complete	0000
ovs	SO - Overflow with signed operands	0001
uns	SU - Underflow with signed operands	0010
ovu	UO - Overflow with unsigned operands	0011
unu	UU - Underflow with unsigned operands	0100
sgn	CS - Sign of result	0101
zero	RZ - Result is zero	0110
mloss	ML - Multiply precision loss	0111
mover	MO - Multiply overflow	1000
all1	MM - Result of mask operation is FFFFh	1001
all0	MN - Result of mask operation is 0000h	1010
aritl	false if add/sub saturation is enabled, true otherwise (see stal instruction)	1011
arith	false if logic is set to two's complement, true if logic is set to positive numbers only (see stal instruction)	1100
carry	C - Carry	1101
conv	CS - Conversion sign	1110
csh	SB - Carry on shift operation	1111

Table 46. jarr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	BitSel				Dest				

Source code example:

```
*### Do 0x0800 & IRQ status register and jump to a label if results equal 0####
irq_and:        cp irq r0;              * Save the irq register into r0 (for this example irq = 0x400 due to a sw interrupt 1)
                ldirh 08h rst;          * Load immediate register ir MSB with 0x08 and reset the LSB -> IR = 0x0800
                and r0;                 * Operation does ir & r0 = 0x0800 & 0x0400 = 0x0000 and save this results in r0
                jarr results_zero all0; * if the results = 0 => sw interrupt was sw 1=> go to results_zero label
results_zero: ### Add code here ###
```

jcrf

Description:

Assembler syntax: `jcrf op1 CrSel Pol;`

Configures the jump to absolute location on control register condition.

If the condition defined by the CrSel operand is satisfied according to the polarity Pol, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- CrSel – Operand defines the control register condition (Ctrl_reg_uc0 (101h and 121h) and Ctrl_reg_uc1 (102h and 122h) registers) that triggers the jump

Operand label	Operand description	Operand binary value
b0	Control register bit 0 (LSB)	0000
b1	Control register bit 1	0001
b2	Control register bit 2	0010
b3	Control register bit 3	0011
b4	Control register bit 4	0100
b5	Control register bit 5	0101
b6	Control register bit 6	0110
b7	Control register bit 7	0111
b8	Control register bit 8	1000
b9	Control register bit 9	1001
b10	Control register bit 10	1010
b11	Control register bit 11	1011
b12	Control register bit 12	1100
b13	Control register bit 13	1101
b14	Control register bit 14	1110
b15	Control register bit 15 (MSB)	1111

Pol – Operand defines the active polarity for the selected bit

Operand label	Operand description	Operand binary value
low	Active condition if the selected bit is '0'	0
high	Active condition if the selected bit is '1'	1

Table 47. jcrf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	Pol	CrSel				op1	0	1	0	0

jcrr

Description:

Configures the jump to relative location on control register condition.

If the condition defined by the CrSel operand is satisfied according to the polarity Pol, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `jcrr Dest CrSel Pol;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}.
- CrSel – Operand defines the control register condition (Ctrl_reg_uc0 (101h, 121h) and Ctrl_reg_uc1 (102h, 122h) registers) that triggers the jump.

Operand label	Operand description	Operand binary value
b0	Control register bit 0 (LSB)	0000
b1	Control register bit 1	0001
b2	Control register bit 2	0010
b3	Control register bit 3	0011
b4	Control register bit 4	0100
b5	Control register bit 5	0101
b6	Control register bit 6	0110
b7	Control register bit 7	0111
b8	Control register bit 8	1000
b9	Control register bit 9	1001
b10	Control register bit 10	1010
b11	Control register bit 11	1011
b12	Control register bit 12	1100
b13	Control register bit 13	1101
b14	Control register bit 14	1110
b15	Control register bit 15 (MSB)	1111

Pol – Operand defines the active polarity for the selected bit

Operand label	Operand description	Operand binary value
low	Active condition if the selected bit is '0'	0
high	Active condition if the selected bit is '1'	1

Table 48. jcrr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	Pol	CrSel				Dest				

jfbkf

Description:

Configures the jump to absolute location on feedback condition.

If the condition defined by the SelFbk operand is satisfied according to the polarity Pol, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `jfbkf op1 SelFbk Pol;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- SelFbk – Operand defines the feedback signal condition

Operand label	Operand description	Operand binary value
hs1v	High-side predriver 1 V_{DS} feedback	000000
hs1s	High-side predriver 1 V_{SRC} feedback	000001
hs2v	High-side predriver 2 V_{DS} feedback	000010
hs2s	High-side predriver 2 V_{SRC} feedback	000011
hs3v	High-side predriver 3 V_{DS} feedback	000100
hs3s	High-side predriver 3 V_{SRC} feedback	000101
hs4v	High-side predriver 4 V_{DS} feedback	000110
hs4s	High-side predriver 4 V_{SRC} feedback	000111
hs5v	High-side predriver 5 V_{DS} feedback	001000
hs5s	High-side predriver 5 V_{SRC} feedback	001001
ls1v	Low-side predriver 1 V_{DS} feedback	001110
ls2v	Low-side predriver 2 V_{DS} feedback	001111
ls3v	Low-side predriver 3 V_{DS} feedback	010000
ls4v	Low-side predriver 4 V_{DS} feedback	010001
ls5v	Low-side predriver 5 V_{DS} feedback	010010
ls6v	Low-side predriver 6 V_{DS} feedback	010011

- Pol – Defines the active polarity for the selected feedback signal

Operand label	Operand description	Operand binary value
low	Active condition if the selected bit is '0'	0
high	Active condition if the selected bit is '1'	1

Table 49. jfbkf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	SelFbk				Pol	op1	0	1	0	0

jfbkr

Description:

Configures the jump to relative location on feedback condition.

If the condition defined by the SelFbk operand is satisfied according to the polarity Pol, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `jfbkr Dest SelFbk Pol;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}.
- SelFbk – Operand defines the feedback signal condition

Operand label	Operand description	Operand binary value
hs1v	High-side predriver 1 V_{DS} feedback	0000
hs1s	High-side predriver 1 V_{SRC} feedback	0001
hs2v	High-side predriver 2 V_{DS} feedback	0010
hs2s	High-side predriver 2 V_{SRC} feedback	0011
hs3v	High-side predriver 3 V_{DS} feedback	0100
hs3s	High-side predriver 3 V_{SRC} feedback	0101
hs4v	High-side predriver 4 V_{DS} feedback	0110
hs4s	High-side predriver 4 V_{SRC} feedback	0111
hs5v	High-side predriver 5 V_{DS} feedback	1000
hs5s	High-side predriver 5 V_{SRC} feedback	1001
ls1v	Low-side predriver 1 V_{DS} feedback	1010
ls2v	Low-side predriver 2 V_{DS} feedback	1011
ls3v	Low-side predriver 3 V_{DS} feedback	1100
ls4v	Low-side predriver 4 V_{DS} feedback	1101
ls5v	Low-side predriver 5 V_{DS} feedback	1110
ls6v	Low-side predriver 6 V_{DS} feedback	1111

Pol – Operand defines the active polarity for the selected bit

Operand label	Operand description	Operand binary value
low	Active condition if the selected bit is '0'	0
high	Active condition if the selected bit is '1'	1

Table 50. jfbkr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	SelFbk				Pol	Dest				

jmpf

Description:

Configures the unconditional jump.

The destination address defined in one of the jump registers defined by the operand op1. The destination address is any of the absolute Code RAM location.

Assembler syntax: `jmpf op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"

Table 51. jmpf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	1	0	1	0	op1	1	0	1

```
*### Jump far to the label eoinj0 ####

init0:      ldjr1 eoinj0;          * Load the eoinj label address to jr1 for a far jump
Convert:    jmpf jr1;              * jump to jr1 which eoinj0
            .....x15 instruction lines.....  *If there are less than 15 lines between the jump and the label address jump far is not required, jump relative is enough
eoinj0:     stos off off off;      * Turn Off all outputs
```

jmpr

Description:

Configures the unconditional jump to relative location.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `jmpr Dest;`

Operands:

- Dest– Operand defines the 5-bit relative destination address in the range of {–16, 15}.

Table 52. jmpr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	1	0	0	0	Dest				

jocf

Description:

Configures the jump to absolute location on condition.

If the condition defined by the Cond operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The feedback from current measurement 5 and 6 can not be checked at the same time by the same microcore (refer to register Dac_rxtx_cr_config (112h and 132h)). The feedback from current measurement 4 can only be checked if this channel is activated via the flags_source (1A3h) register. If the channel is activated, flag 12 can not be checked anymore.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `jocf op1 Cond;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- Cond – Operand defines the condition to be satisfied to enable the jump far

Operand label	Operand description
_f0	Flag 0 low
_f1	Flag 1 low
_f2	Flag 2 low
_f3	Flag 3 low
_f4	Flag 4 low
_f5	Flag 5 low
_f6	Flag 6 low
_f7	Flag 7 low
_f8	Flag 8 low
_f9	Flag 9 low
_f10	Flag 10 low
_f11	Flag 11 low
_f12	Flag 12 low
_f13	Flag 13 low
_f14	Flag 14 low
_f15	Flag 15 low
f0	Flag 0 high
f1	Flag 1 high
f2	Flag 2 high
f3	Flag 3 high
f4	Flag 4 high

Operand label	Operand description
f5	Flag 5 high
f6	Flag 6 high
f7	Flag 7 high
f8	Flag 8 high
f9	Flag 9 high
f10	Flag 10 high
f11	Flag 11 high
f12 / cur4	Flag 12 high / Current feedback 4 high
f13	Flag 13 high
f14	Flag 14 high
f15	Flag 15 high
tc1	Terminal count 1
tc2	Terminal count 2
tc3	Terminal count 3
tc4	Terminal count 4
_start	Start low
start	Start high
_sc1v	Shortcut1 V_{DS} feedback low
_sc2v	Shortcut2 V_{DS} feedback low
_sc3v	Shortcut3 V_{DS} feedback low
_sc1s	Shortcut1 source feedback low
_sc2s	Shortcut2 source feedback low
_sc3s	Shortcut3 source feedback low
sc1v	Shortcut1 V_{DS} feedback high
sc2v	Shortcut2 V_{DS} feedback high
sc3v	Shortcut3 V_{DS} feedback high
opd	Instruction request to ALU executed
vb	Boost voltage high
_vb	Boost voltage low
cur1	Current feedback 1 high
cur2	Current feedback 2 high
cur3	Current feedback 3 high
cur4l	Current feedback 5/6l high
cur4h	Current feedback 5/6h high
cur4n	Current feedback 5/6n high
_cur1	Current feedback 1 low
_cur2	Current feedback 2 low

Operand label	Operand description
_cur3	Current feedback 3 low
_cur4l	Current feedback 5/6l low
_cur4h	Current feedback 5/6h low
_cur4n	Current feedback 5/6n low
ocur	Own current feedback high
_ocur	Own current feedback low

Table 53. jocf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	Cond						op1	0	0	0	0

jocr

Description:

Configures the jump to relative location on condition.

If the condition defined by the Cond operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}. It is possible to supply a label for this field.

Assembler syntax: `jocr Dest Cond;`

Operands:

- Dest– Operand defines the 5-bit relative destination address in the range of {–16, 15}.
- Cond– Operand defines the condition to be satisfied to enable the relative jump

Operand label	Operand description
_f0	Flag 0 low
_f1	Flag 1 low
_f2	Flag 2 low
_f3	Flag 3 low
_f4	Flag 4 low
_f5	Flag 5 low
_f6	Flag 6 low
_f7	Flag 7 low
_f8	Flag 8 low
_f9	Flag 9 low
_f10	Flag 10 low
_f11	Flag 11 low
_f12	Flag12 low
_f13	Flag 13 low
_f14	Flag 14 low
_f15	Flag 15 low
f0	Flag 0 high
f1	Flag 1 high
f2	Flag 2 high
f3	Flag 3 high
f4	Flag 4 high
f5	Flag 5 high
f6	Flag 6 high
f7	Flag 7 high

Operand label	Operand description
f8	Flag 8 high
f9	Flag9 high
f10	Flag10 high
f11	Flag11 high
f12	Flag12 high
f13	Flag13 high
f14	Flag14 high
f15	Flag15 high
tc1	Terminal count 1
tc2	Terminal count 2
tc3	Terminal count 3
tc4	Terminal count 4
_start	Startlow
start	Starthigh
_sc1v	Shortcut1 V_{DS} feedback low
_sc2v	Shortcut2 V_{DS} feedback low
_sc3v	Shortcut3 V_{DS} feedback low
_sc1s	Shortcut1 source feedback low
_sc2s	Shortcut2 source feedback low
_sc3s	Shortcut3 source feedback low
sc1v	Shortcut1 V_{DS} feedback high
sc2v	Shortcut2 V_{DS} feedback high
sc3v	Shortcut3 V_{DS} feedback high
opd	Instruction request to ALU executed
vb	Boost voltage high
_vb	Boost voltage low
cur1	Current feedback 1 high
cur2	Current feedback 2 high
cur3	Current feedback 3 high
cur4l	Current feedback 5/6l high
cur4h	Current feedback 5/6h high
cur4n	Current feedback 5/6n high
_cur1	Current feedback 1 low
_cur2	Current feedback 2 low
_cur3	Current feedback 3 low
_cur4l	Current feedback 5/6l low
_cur4h	Current feedback 5/6h low

Operand label	Operand description
_cur4n	Current feedback 5/6n low
ocur	Own current feedback high
_ocur	Own current feedback low

Table 54. jocr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	Cond						Dest				

joidf

Description:

Configures the jump to absolute location on microcore identifier condition.

If the condition defined by the UcSel operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `joidf op1 UcSel;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- UcSel – Operand defines the microcore identifier condition

Operand label	Operand description
uc0	The microcore 0 is the current microcore
uc1	The microcore 1 is the current microcore

Table 55. joidf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	1	0	0	UcSel	op1	1	0	1

joidr

Description:

Configures the jump to a relative location on condition.

If the condition defined by the UcSel operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `joidr Dest UcSel;`

Operands:

- Dest– Operand defines the 5-bit relative destination address in the range of {–16, 15}
- UcSel– Operand defines the microcore identifier condition

Operand label	Operand description
uc0	The microcore 0 is the current microcore
uc1	The microcore 1 is the current microcore

Table 56. joidr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	1	0	1	UcSel	Dest				

joslf

Description:

Configures the jump to absolute location on condition.

If the condition defined by the StSel operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `joslf op1 StSel;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- StSel – Operand defines the start condition to be satisfied to enable the jump far

Operand label	Operand description
none	No start latched
start1	Start 1 latched
start2	Start 2 latched
start12	Start 1,2 latched
start3	Start 3 latched
start13	Start 1,3 latched
start23	Start 2,3 latched
start123	Start 1,2,3 latched
start4	Start 4 latched
start14	Start 1,4 latched
start24	Start 2,4 latched
start124	Start 1,2,4 latched
start34	Start 3,4 latched
start134	Start 1,3,4 latched
start234	Start 2,3,4 latched
start1234	Start 1,2,3,4 latched
start5	Start 5 latched
start15	start 1,5 latched
start25	start 2,5 latched
start125	start 1,2,5 latched
start35	start 3,5 latched
start135	start 1,3,5 latched
start235	start 2,3,5 latched
start1235	start 1,2,3,5 latched
start45	start 4,5 latched

Operand label	Operand description
start145	start 1,4,5 latched
start245	start 2,4,5 latched
start1245	start 1,2,4,5 latched
start345	start 3,4,5 latched
start1345	start 1,3,4,5 latched
start2345	start 2,3,4,5 latched
start12345	start 1,2,3,4,5 latched
start6	start 6 latched
start16	start 1,6 latched
start26	start 2,6 latched
start126	start 1,2,6 latched
start36	start 3,6 latched
start136	start 1,3,6 latched
start236	start 2,3,6 latched
start1236	start 1,2,3,6 latched
start46	start 4,6 latched
start146	start 1,4,6 latched
start246	start 2,4,6 latched
start1246	start 1,2,4,6 latched
start346	start 3,4,6 latched
start1346	start 1,3,4,6 latched
start2346	start 2,3,4,6 latched
start12346	start 1,2,3,4,6 latched
start56	start 5,6 latched
start156	start 1,5,6 latched
start256	start 2,5,6 latched
start1256	start 1,2,5,6 latched
start356	start 3,5,6 latched
start1356	start 1,3,5,6 latched
start2356	start 2,3,5,6 latched
start12356	start 1,2,3,5,6 latched
start456	start 4,5,6 latched
start1456	start 1,4,5,6 latched
start2456	start 2,4,5,6 latched
start12456	start 1,2,4,5,6 latched
start3456	start 3,4,5,6 latched
start13456	start 1,3,4,5,6 latched

Operand label	Operand description
start23456	start 2,3,4,5,6 latched
start123456	start 1,2,3,4,5,6 latched

Table 57. joslf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	StSel						op1	0	0	0	1

joslr

Description:

Configures the jump to relative location on condition.

If the condition defined by the StSel operand is satisfied, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `joslr Dest StSel;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}
- StSel – Operand defines the start condition to be satisfied to enable the jump far

Operand label	Operand description
none	No start latched
start1	Start 1 latched
start2	Start 2 latched
start12	Start 1,2 latched
start3	Start 3 latched
start13	Start 1,3 latched
start23	Start 2,3 latched
start123	Start 1,2,3 latched
start4	Start 4 latched
start14	Start 1,4 latched
start24	Start 2,4 latched
start124	Start 1,2,4 latched
start34	Start 3,4 latched
start134	Start 1,3,4 latched
start234	Start 2,3,4 latched
start1234	Start 1,2,3,4 latched
start5	Start 5 latched
start15	start 1,5 latched
start25	start 2,5 latched
start125	start 1,2,5 latched
start35	start 3,5 latched
start135	start 1,3,5 latched
start235	start 2,3,5 latched
start1235	start 1,2,3,5 latched

Operand label	Operand description
start45	start 4,5 latched
start145	start 1,4,5 latched
start245	start 2,4,5 latched
start1245	start 1,2,4,5 latched
start345	start 3,4,5 latched
start1345	start 1,3,4,5 latched
start2345	start 2,3,4,5 latched
start12345	start 1,2,3,4,5 latched
start6	start 6 latched
start16	start 1,6 latched
start26	start 2,6 latched
start126	start 1,2,6 latched
start36	start 3,6 latched
start136	start 1,3,6 latched
start236	start 2,3,6 latched
start1236	start 1,2,3,6 latched
start46	start 4,6 latched
start146	start 1,4,6 latched
start246	start 2,4,6 latched
start1246	start 1,2,4,6 latched
start346	start 3,4,6 latched
start1346	start 1,3,4,6 latched
start2346	start 2,3,4,6 latched
start12346	start 1,2,3,4,6 latched
start56	start 5,6 latched
start156	start 1,5,6 latched
start256	start 2,5,6 latched
start1256	start 1,2,5,6 latched
start356	start 3,5,6 latched
start1356	start 1,3,5,6 latched
start2356	start 2,3,5,6 latched
start12356	start 1,2,3,5,6 latched
start456	start 4,5,6 latched
start1456	start 1,4,5,6 latched
start2456	start 2,4,5,6 latched
start12456	start 1,2,4,5,6 latched
start3456	start 3,4,5,6 latched

Operand label	Operand description
start13456	start 1,3,4,5,6 latched
start23456	start 2,3,4,5,6 latched
start123456	start 1,2,3,4,5,6 latched

Table 58. joslr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	StSel						Dest				

jsrf

Description:

Configures the jump to absolute location on status register condition.

If the condition defined by the SrSel operand is satisfied according to the polarity Pol, the program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `jsrf op1 SrSel Pol;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"
- SrSel – Operand defines the status register condition (Status_reg_uc0 (105h, 125h) and Status_reg_uc1 (106h, 126h)) that triggers the jump

Operand label	Operand description
b0	Status register bit 0 (LSB)
b1	Status register bit 1
b2	Status register bit 2
b3	Status register bit 3
b4	Status register bit 4
b5	Status register bit 5
b6	Status register bit 6
b7	Status register bit 7
b8	Status register bit 8
b9	Status register bit 9
b10	Status register bit 10
b11	Status register bit 11
b12	Status register bit 12
b13	Status register bit 13
b14	Status register bit 14
b15	Status register bit 15 (MSB)

Pol – Operand defines the active polarity for the selected bit

Operand label	Operand description
low	Active condition if the selected bit is '0'
high	Active condition if the selected bit is '1'

Table 59. jsrf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	Pol	SrSel				op1	0	1	0	1

jsrr

Description:

Configures the jump to the relative location of the status register condition.

If the condition defined by the SrSel operand is satisfied according to the polarity Pol, the program counter (uPC) is handled such as the next executed instruction is relative destination address.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `jsrr Dest SrSel Pol;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}
- SrSel – Operand defines the status register condition (Status_reg_uc0 (105h, 125h) and Status_reg_uc1 (106h, 126h) registers) that triggers the jump

Operand label	Operand description
b0	Status register bit 0 (LSB)
b1	Status register bit 1
b2	Status register bit 2
b3	Status register bit 3
b4	Status register bit 4
b5	Status register bit 5
b6	Status register bit 6
b7	Status register bit 7
b8	Status register bit 8
b9	Status register bit 9
b10	Status register bit 10
b11	Status register bit 11
b12	Status register bit 12
b13	Status register bit 13
b14	Status register bit 14
b15	Status register bit 15 (MSB)

Pol – Operand defines the active polarity for the selected bit

Operand label	Operand description
low	Active condition if the selected bit is '0'
high	Active condition if the selected bit is '1'

Table 60. jsrr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	Pol	SSel				Dest				

jtsf

Description:

Configures the jump on subroutine to absolute location

The program counter (uPC) is handled such as the next executed instruction is located into the destination address contained in one of the jump registers.

When jump to subroutine is called, the current program counter value (uPC) is stored in the auxiliary register (aux) to handle the end-of-subroutine return.

The destination address defined by the op1 register is any of the absolute Code RAM location.

Assembler syntax: `jtsf op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.4 "JpReg subset"

Table 61. jtsf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	1	0	1	1	op1	1	0	1

jtsr

Description:

Configures the jump to subroutine to relative location on condition.

When jump to subroutine is called, the current program counter value (uPC) is stored into the auxiliary register (aux) to handle end-of-subroutine return.

The jump is relative to the instruction Code RAM location. The destination address is the actual instruction Code RAM location added to the Dest operand value. This 5-bit value is a two's complemented number. The MSB is the sign. So Dest operand value is in the range of {–16, 15}.

Assembler syntax: `jtsr Dest;`

Operands:

- Dest – Operand defines the 5-bit relative destination address in the range of {–16, 15}.

Table 62. jtsr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	1	0	0	1	Dest				

ldca

Description:

Loads one of the four end-of-count registers (eoc1, eoc2, eoc3, eoc4) defined by the operand with a value stored in a ALU register op1 and sets the outputs defined by the shortcut Sh1 and Sh2.

Assembler syntax: `ldca Rst Sh1 Sh2 op1 Eoc;`

Operands:

- Rst – Operand (Boolean) defines if the selected counter value must be reset to zero or must be unchanged.

Operand label	Operand description
_rst	The counter value is maintained, only the end-of-counter is modified
rst	The counter value is reset to zero and starts to count from zero

Sh1, Sh2 – Operands set the first and second shortcuts related to the corresponding outputs. The output shortcuts are defined using the dfsct instruction.

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Shortcut (1 or 2) turns OFF
on	Shortcut (1 or 2) turns ON
toggle	Reverse the previous setting

op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Eoc – Operand defines the end-of-count targeted among the four counters available.

Operand label	Operand description
c1	Register eoc1
c2	Register eoc2
c3	Register eoc3
c4	Register eoc4

Table 63. ldca instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	Rst	Sh1		Sh2		Eoc		op1			1	0	0

ldcd

Description:

Loads one of the four end-of-count registers (eoc1, eoc2, eoc3, eoc4) with a value stored in the 6-bit Data RAM address Dram and sets the outputs defined by the shortcut Sh1 and Sh2.

The operand Dram can be identified with a univocal label. The compiler automatically substitutes the 'define' label (if used) with the suitable Data RAM address.

The Data RAM address is accessed according to the Boolean operand Ofs using the Immediate addressing mode (IM), Indexed addressing mode (XM). In that case address base is added to the address Dram. The address base is set using the stab instructions.

Assembler syntax: `ldcd Rst Ofs Sh1 Sh2 Dram Eoc;`

Operands:

- Rst – Operand (Boolean) defines if the selected counter value must be reset to zero or must be unchanged.

Operand label	Operand description
_rst	The counter value is maintained, only the end-of-count is modified
rst	The counter value is reset to zero and start to count from zero

Ofs – Operands set Data RAM addressing mode

Operand label	Operand description
_ofs	Data RAM immediate addressing mode (IM)
ofs	Data RAM indexed addressing mode (XM)

Sh1, Sh2 – Operands set the first and second shortcuts related to the corresponding outputs. The output shortcuts are defined using the dfsct instruction.

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Shortcut (1 or 2) turns OFF
on	Shortcut (1 or 2) turns ON
toggle	Reverse the previous setting

Dram – Operand defines the 6-bit DRAM address

Eoc – Operand defines the end-of-count targeted among the four counters available.

Operand label	Operand description
c1	Register eoc1
c2	Register eoc2
c3	Register eoc3
c4	Register eoc4

Table 64. Idcd instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	Ofs	Rst	Sh1		Sh2		Eoc		Dram					

ldirh

Description:

Add two ALU registers and place the result in one of the ALU registers

Loads the Value8 data in the 8-MSB of the immediate register (ir).

Assembler syntax: `ldirh Value8 RstL;`

- Value8 – Operand defines the 8-bit value to be loaded into the 8-MSB of the immediate register
- RstL – Operand (Boolean) defines if set to zero the low-byte (7:0) of ir register

Operand label	Operand description
_rst	No change on the ir[7:0]
rst	Set the Zero the ir[7:0]

Table 65. ldirh instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	RstL	Value8								1	1

ldirl

Description:

Add two ALU registers and place the result in one of the ALU registers

Loads the Value8 data in the 8-LSB of the immediate register (ir).

Assembler syntax: `ldirl Value8 RstH;`

Operands:

- Value8– Operand defines the 8-bit value to be loaded into the 8-MSB of the immediate register
- RstH–Operand (Boolean) defines if set to zero the high-byte (15:8) of ir register

Operand label	Operand description
_rst	No change on the ir[15:8]
rst	Set the Zero on the ir[15:8]

Table 66. ldirl instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	RstH	Value8								1	0

ldjr1

Description:

Loads the Value10 data in the 16-bit jump register 1 (jr1).

The operand Value10 can be replaced by a label. The compiler automatically substitutes the label (if used) with the defined value.

Assembler syntax: `ldjr1 Value10;`

Operands:

- Value10 – Operand defines the 10-bit value to be loading into the jump register 1

Table 67. ldjr1 instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Value10										0	0

Source code example:

```
*### Jump far to the label eoinj0 ####

init0:      ldjr1 eoinj0;          * Load the eoinj label address to jr1 for a far jump
Convert:    jmpf jr1;              * jump to jr1 which eoinj0
            .....x15 instruction lines.....  *If there are less than 15 lines between the jump and the label address jump far is not required, jump relative is enough
eoinj0:     stos off off off;      * Turn Off all outputs
```

ldjr2

Description:

Loads the Value10 data in the 16-bit jump register 2 (jr2).

The operand Value10 can be replaced by a label. The compiler automatically substitutes the label (if used) with the defined value.

Assembler syntax: `ldjr2 Value10;`

Operands:

- Value10 – Operand defines the 10-bit value to be loading into the jump register 2

Table 68. ldjr2 instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Value10										0	0

load

Description:

Loads the data from the Data RAM at the address defined by the Dram operand to the op1 register.

The operand Dram can be identified with a univocal label. The compiler automatically substitutes the 'define' label (if used) with the suitable Data RAM address.

The Data RAM address is accessed according to the Boolean operand Ofs using the Immediate addressing mode (IM), Indexed addressing mode (XM). In that case, address base is added the address Dram. The address base is set using the stab instructions.

Assembler syntax: `load Dram op1 Ofs;`

Operands:

- Dram – Operand defines the 6-bit Data RAM address
- op1 – One of the registers listed in the operand Section 3.1.3 "UcReg subset"
- Ofs – Operands set data RAM addressing mode

Operand label	Operand description
_ofs	Data RAM immediate addressing mode (IM)
ofs	Data RAM indexed addressing mode (XM)

Table 69. load instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	Dram						op1					Ofs	1	0

mul

Description:

Multiplies the value contained in the op1 register with the value contained in op2 register and places the result in the reg32 register. The reg32 register is the concatenation of the multiplication result registers mh and ml:

mh contains the 16-MSB ml contains the 16-MSB

The multiplication requires 32 ck clock cycles to be completed.

Operation: (Source1) x (Source2) → (Destination)

Assembler syntax: `mul op1 op2;`

Operands:

- op1– One of the registers listed in the operand Section 3.1.2 "AluGprIrReg subset"
- op2– One of the registers listed in the operand Section 3.1.2 "AluGprIrReg subset"

Condition register:

- MO – Multiplication shift overflow
- ML – Multiplication shift precision loss
- OD – Operation complete

Table 70. mul instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	1	1	0	op2			op1		

```
*#### Multiply MUL1 by MUL2 and store MSBs result in r1 register and LSBs result in r0 register ########
#define MUL1 0;   * The boost phase current value is stored in Data RAM address 0
#define MUL2 1;   * The peak phase current value is stored in Data RAM address 1

multiplication:   load MUL1 r0 _ofs;          * Load MUL1 value from DRAMaddress 1 into the r0 register
                  load MUL2 r1 _ofs;          * Load MUL2 value from DRAMaddress 2 into the r1 register
                  mul r0 r1;                  * Multiply MUL1 by MUL2
                  cwer MulDone opd row1;      * Create a Wait entry until the operation is finished (required because mul takes 32ck cycles)
                  wait row1;                  * Wait here until operation is done then go to MulDone label

MulDone:          cp mh r0;                   * Save MSB results into r0 register
                  cp ml r1;                   * Save LSB results into r1 register
```

muli

Description:

Multiplies the value contained in the op1 register with the immediate value Imm and places the result in the reg32 register. The reg32 register is the concatenation of the multiplication result registers mh and ml:

mh contains the 16-MSB

ml contains the 16-LSB

The multiplication requires 32 ck clock cycles to be completed.

Assembler syntax: `muli op1 Imm;`

Operands:

- op1 – One of the registers listed in the operand AluGprIrReg subset
- Imm – The Imm 4-bit immediate data register

Condition register:

- MO – Multiplication shift overflow
- ML – Multiplication shift precision loss
- OD – Operation complete

Table 71. muli instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	1	0	Imm				op1		

not

Description:

Inverts each bit of the op1 register and places the result in the op1 register.

Operation: (Source) \ → (Source)

Assembler syntax: `not op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- MN– Mask result is 0000h
- MM- Mask result is FFFFh

Table 72. not instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	1	0	1	1	op1		

or

Description:

Applies the OR-mask stored in the Immediate Register (ir) to the op1 register and places the result in the op1 register.

Operation: (Source) (+) Immediate register → (Source)

Assembler syntax: `or op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- MN– Mask result is 0000h
- MM- Mask result is FFFFh

Table 73. or instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	1	0	0	0	op1		

rdspi

Description:

Requests an SPI backdoor read.

The address must previously be defined in the SPI address register spi_add.

The rdspi instruction requires 2 ck cycle to complete operation. The SPI address register must not be changed on the following instruction, otherwise the operation fails and the read data is dummy.

Assembler syntax: `rdspi;`

Table 74. rdspi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0	0	0	0	0

```
*#### Read register 125h using SPI backdoor ########
SPI_Init:     slsa ir;              * Configure SPI accesses to use 'ir' as SPI address register
Ld_Add:       ldirh 01h _rst;       * Load 01h in the ir MSB register
              ldirl 25h _rst;       * Load 25h in the ir LSB register -> ir = 125h
SPI_Read:     rdspi;                * Read SPI using ir for address (125h)
Save_data:    cp spi_data r0;       * Copy the register 125h value (spi_data) into r0
```

reqi

Description:

Requests a software interrupt

At the reqi instruction execution, the Code RAM address currently executed is stored in the interrupt return register corresponding to the 10 LSB of the uc0_irq_status register (10Fh, 12Fh) and for uc1_irq_status (110h, 130h).

By default, the return address of an interrupt is the line where the code was interrupted. In the case of a software interrupt, the return address is the address where the code was interrupted + 1.

A software interrupt must not be interrupted.

Assembler syntax: `reqi id;`

Operands:

- Id – Operand defines the 2-bit software interrupt request identifier.

Table 75. reqi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	0	1	0	Id	

rfs

Description:

Ends a subroutine.

To continue the code execution, the program counter (uPC) is loaded with the content of the auxiliary register (aux) automatically updated when the subroutine was called with the instructions jtsf and jtsr.

Assembler syntax: `rfs;`

Table 76. rfs instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0

rstreg

Description:

Resets single or multiple registers defined by the TgtBit operand. The instruction reset bits issued from SPI registers including:

control register ctrl_reg_uc0 (101h, 121h) and ctrl_reg_uc1 (102, 122)

status register status_reg_uc0 (105h, 125h) and status_reg_uc1 (106h, 126h) automatic diagnosis

register Err_ucXchY (162h to 169h)

Assembler syntax: `rstreg TgtBit;`

Operands:

- TgtBit – Operands defines the registers to be reset.

Operand label	Operand description
sr	Reset status bits of the status registers
cr	Reset control register
sr_diag_halt	Reset status bits, automatic diagnosis register and re-enables the possibility to generate automatic diagnosis interrupts
all	Reset status bits, control register, automatic diagnosis register and re-enables the possibility to generate automatic diagnosis interrupts
diag_halt	Reset automatic diagnosis register and re-enables the possibility to generate automatic diagnosis interrupts
sr_cr	Reset status bits and control register
sr_halt	Reset status bits and re-enables the possibility to generate automatic diagnosis interrupts
halt	Re-enables the possibility to generate automatic diagnosis interrupts

Table 77. rstreg instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0	1	TgtBit		

rstsl

Description:

Resets the Start_latch_ucx register.

This instruction is active only if the Smart Latch mode is enabled. The smart mode register can be activated by setting the bits

smart_start_uc0 and smart_start_uc1 of the Start_config_reg_Part2 registers (104h, 124h).

Assembler syntax: `rstsl;`

Table 78. rstsl instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	1

sh32l

Description:

Shifts the reg32 register left. The shift is single or multiple according to the op1 register value (factor). The reg32 register is the concatenation of the multiplication result registers mh and ml:

mh contains the 16-MSB ml contains the 16-LSB

To be completed, the shift operation requires a number of ck clock cycles corresponding to the op1 register value.

Operation: (Source1) << factor → (Source)

Assembler syntax: `sh32l op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 79. sh32l instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	0	1	op1			1	0	1

sh32li

Description:

Shifts the reg32 register left. The shift is single or multiple according to the immediate value (factor). The reg32 register is the concatenation of the multiplication result registers mh and ml:

mh contains the 16-MSB ml contains the 16-LSB

To be completed, the shift operation requires a number of ck clock cycles corresponding to the immediate value.

Operation: (Source1) << Immediate value → (Source)

Assembler syntax: `sh32li Imm;`

Operands:

- Imm – The Imm 4-bit immediate data register

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 80. sh32li instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	0	1	Imm				1	0	1

sh32r

Description:

Shifts the reg32 register right. The right shift is single or multiple according to the op1 register value (factor). The reg32 register is the concatenation of the multiplication result registers mh and ml:

mh contains the 16-MSB ml contains the 16-LSB

To be completed, the shift operation requires a number of ck clock cycles corresponding to the op1 register value.

Operation: (Source1) << factor → (Source)

Assembler syntax: `sh32r op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- ML – Multiplication shift precision loss

Table 81. sh32r instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	0	0	op1			1	0	1

sh32ri

Description:

Shifts the reg32 register right. The right shift is single or multiple according to the immediate value. The reg32 register is the concatenation of the multiplication result registers mh and ml:

mh contains the 16-MSB ml contains the 16-LSB

To be completed, the shift operation requires a number of ck clock cycles corresponding to the immediate value.

Operation: (Source1) << Immediate value → (Source)

Assembler syntax: `sh32ri Imm;`

Operands:

- Imm – The Imm 4-bit immediate data register

Condition register:

- SB – Shift out bit
- ML – Multiplication shift precision loss

Table 82. sh32ri instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	0	0	Imm				1	0	1

shl

Description:

Shifts the op1 register left. The shift is single or multiple according to the op2 register value (factor).

To be completed, the shift operation requires a number of ck clock cycles corresponding to the op2 register value.

Operation: (Source1) << factor → (Source)

Assembler syntax: `shl op1 op2;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 83. shl instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	0	op2			op1		

shl8

Description:

Shifts the op1 register 8 positions left.

To be completed, the shift operation requires one ck clock cycles.

Operation: (Source1) << 8 → (Source)

Assembler syntax: `shl8 op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 84. shl8 instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	1	1	1	1	op1		

shli

Description:

Shift the op1 register left. The shift is single or multiple according to the immediate value Imm.

To be completed, the shift operation requires a number of ck clock cycles corresponding to the immediate value Imm.

Operation: (Source1) << Immediate value → (Source)

Assembler syntax: `shl op1 Imm;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Imm – The Imm 4-bit immediate data register

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 85. shli instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	1	0	Imm				op1		

shls

Description:

Shift the op1 register left. The shift is single or multiple according to the op2 register value (factor).

The op1 register is handled as a two's complement number. The MBS (sign bit) is unchanged during the shift operation. To be completed, the shift operation requires a number of ck clock cycles corresponding to the op2 register value.

Operation: (Source1) << factor → (Source)

Assembler syntax: `shls op1 op2;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 86. shls instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	0	0	op2			op1		

shlsi

Description:

Shifts the op1 register left. The shift is single or multiple according to the immediate value Imm.

The op1 register is handled as a two's complement number. The MBS (sign bit) is unchanged during the shift operation. To be completed, the shift operation requires a number of ck clock cycles corresponding to the immediate value Imm.

Operation: (Source1) << Immediate value → (Source)

Assembler syntax: `shls op1 Imm;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Imm – The Imm 4-bit immediate data register

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 87. shlsi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	0	0	Imm				op1		

shr

Description:

Shift the op1 register right. The shift is single or multiple according to the op2 register value (factor).

To be completed, the shift operation requires a number of ck clock cycles corresponding to the op2 register value.

Operation: (Source1) << factor → (Source)

Assembler syntax: `shr op1 op2;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- ML – Multiplication shift precision loss

Table 88. shr instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	0	1	op2			op1		

Source code example:

```
*### Shift the r3 register by the number of bits in the r2 register and wait until shift is complete ####
shift:      shr r3 r2;              * shift
wait_loop:  jarr done opd;          * jump to label done if shift is finished
            jmpr wait_loop;         * jump back to wait_loop label
done:       *#### Add code here ####
```

shr8

Description:

Shift the op1 register 8 positions right.

To be completed, the shift operation requires one ck clock cycle.

Operation: (Source1) >> 8 → (Source)

Assembler syntax: `shr8 op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- ML – Multiplication shift precision loss

Table 89. shr8 instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	1	1	1	0	op1		

shri

Description:

Shifts the op1 register right. The shift is single or multiple according to the immediate value Imm.

To be completed, the shift operation requires a number of ck clock cycles corresponding to the immediate value Imm.

Operation: (Source1) >> Immediate value → (Source)

Assembler syntax: `shri op1 Imm;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Imm – The Imm 4-bit immediate data register

Condition register:

- SB – Shift out bit
- ML – Multiplication shift precision loss

Table 90. shri instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	0	1	Imm				op1		

shrs

Description:

Shift the op1 register right. The shift is single or multiple according to the op2 register value (factor).

The op1 register is handled as a two's complement number. The MBS (sign bit) is unchanged during the shift operation. To be completed, the shift operation requires a number of ck clock cycles corresponding to the op2 register value.

Operation: (Source1) >> factor → (Source)

Assembler syntax: `shrs op1 op2;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- SB – Shift out bit
- ML – Multiplication shift precision loss

Table 91. shrs instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	1	1	1	op2			op1		

shrsi

Description:

Shifts the op1 register right. The shift is single or multiple according to the immediate value Imm.

The op1 register is handled as a two's complement number. The MBS (sign bit) is unchanged during the shift operation. To be completed, the shift operation requires a number of ck clock cycles corresponding to the immediate value Imm.

Operation: (Source) >> Immediate value → (Source)

Assembler syntax: `shrsi op1 Imm;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Imm – The Imm 4-bit immediate data register

Condition register:

- SB – Shift out bit
- MO – Multiplication shift overflow

Table 92. shrsi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	1	1	Imm				op1		

slab

Description:

Selects the register containing the address (add_base) used in the data RAM Indexed Addressing Mode (XM).

The reset value of SelBase is reg.

Assembler syntax: `slab SelBase;`

Operands:

- SelBase – Operand defines the register to be used to determine the data RAM address base

Operand label	Operand description
reg	Use the dedicated address base add_base register. In this case the address base is defined with the stab instruction.
ir	Use the ALU ir register as address base

Table 93. slab instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	0	0	0	1	0	1	SelBase

Source code example:

```
*### Example 1 ###
*### Use indexed addressing mode and the 'add_base' register to store 33h to address 16d ####
#define Test 0;                          * Define Test to be stored in Data RAM address 0
Set_Add:    slab reg;                    * add_base register selected to offset the address
            stab 10h;                    * set address register to 10h
            ldirl 33h rst;               * set ir LSB to 33h and reset MSB -> ir = 0033h
            store ir Test ofs;           * store ir inside Test + ofs (add_base)= 0 + 10h = 16d, ir is stored in address 16

*### Example 2###
*### Use indexed addressing mode and the 'ir' register to store DDh to address 32d ####
#define Test 0;                          * Define Test to be stored in Data RAM address 0
Set_Add:    slab ir;                     * ir register selected to offset the address
            ldirl 55h _rst;              * set ir LSB to 55h
            cp ir r0;                    * copy ir into r0 -> r0 = 55h
            ldirl 20h rst;               * set ir LSB to 20h and reset MSB -> ir = 0020h, it is used as an offset
            store r0 Test ofs;           * store ir inside Test + ofs (ir)= 0 + 20h = 32d, r0 is stored in address 32
```

slfbk

Description:

Selects the feedback reference for both V_{DS} of the high-side predrivers 2 and 4. In addition, this instruction enables the automatic diagnosis.

This operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

The reset of Ref value is boost.

Assembler syntax: `slfbk Ref Diag;`

Operands:

- Ref – Operand defines the feedback reference for both V_{DS} of the high-side predrivers 2 and 4.

Operand label	Operand description
boost	Both V_{DS} of the high-side predrivers 2 and 4 are referred to boost voltage (VBOOST pin)
bat	Both V_{DS} of the high-side predrivers 2 and 4 are referred to bat voltage (VBATT pin)

- Diag – Operand defines the diagnosis status for both V_{DS}s of the high-side predrivers 2 and 4.

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Automatic diagnosis disabled
on	Automatic diagnosis enabled

Table 94. slfbk instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	1	0	0	0	Ref	Diag	

slfbks

Description:

Selects the feedback for V_{DS} HS1 to HS5 based on shortcut.

In addition, this instruction enables the automatic diagnosis.

This operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

The reset of Ref value is boost.

Assembler syntax: `slfbk Dest Ref Diag;`

Operands:

- Dest – Shortcut used for VDS feedback.

Operand label	Operand description
Sh1	Shortcut 1
Sh2	Shortcut 2
Sh3	Shortcut 3

- Ref – Operand defines the feedback reference for both V_{DS}s of the high-side predrivers 2, 4, and 6.

Operand label	Operand description
boost	Both V_{DS}s of the high-side predrivers 2 and 4 are referred to boost voltage (VBOOST pin)
bat	Both V_{DS}s of the high-side predrivers 2 and 4 are referred to bat voltage (VBATT pin)

- Diag – Operand defines the diagnosis status for both V_{DS}s of the high-side predrivers 2 and 4.

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Automatic diagnosis disabled
on	Automatic diagnosis enabled

Table 95. slfbks instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	1	0	Dest		Ref	Diag	

slsa

Description:

Selects the register containing the address used on SPI read and write instructions (rdspi and wrspi)

The reset values of SelSpi is reg.

Assembler syntax: `slsa SelSpi;`

Operands:

- SelSpi – Operand defines the register containing the SPI address

Operand label	Operand description	Operand binary value
reg	Use the dedicated address register spi_add.	0
ir	Use the ALU ir register as SPI address	1

Table 96. slsa instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	SelSpi

Source code example:

```
* ### Read data using SPI backdoor at SPI address 125h and store result in r0 register ###
SPI_Init:     slsa ir;                  * Configure SPI accesses to use 'ir' for addresses
Ld_Add:       ldirh 01h _rst;           * Load 01h in the ir MSB register
              ldirl 25h _rst;           * Load 25h in the ir LSB register -> ir = 125h
SPI_Read:     rdspi;                    * Read SPI using ir for address (125h)
Save_data:    cp spi_data r0;           * Copy the register 125h value (spi_data) into r0
```

stab

Description:

Loads the address value in the address base register add_base.

The address base register is a 6-bit register containing the address base used in the Data RAM Indexed Addressing Mode (XM).

The operand add_base can be identified with a univocal label. The compiler automatically substitutes the 'define' label (if used) with the suitable address.

Assembler syntax: `stab Add_Base;`

Operands:

- add_base – Operand defines the 6-bit register containing the Address Base.

Table 97. stab instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	0	0	add_base					

Source code example:

```
*### Use indexed addressing mode and the ‘add_base’ register to store 33h to address 16d ####
#define Test 0;                                   * Define Test to be stored in Data RAM address 0
Set_Add:        slab reg;                         * add_base register selected to offset the address
                stab 10h;                         * set address register to 10h
                ldirl 33h rst;                    * set ir LSB to 33h and reset MSB -> ir = 0033h
store ir Test ofs;    * store ir inside Test + ofs (add_base)= 0 + 10h = 16d, ir is stored in address 16h
```

stadc

Description:

Enables or disables the ADC conversion mode on the specified current measurement block. The other channel is selected by SPI register bit (Dac_rxtx_cr_config (112h, 132h, 152h))

The operation is successful only if the microcore has the right to access the related current measurement block. The access right is granted by setting the related bits in the Cur_block_access_partX register (166h, 167h, 168h).

The reset value of AdcMode is off.

Assembler syntax: `stadc AdcMode DacTarget;`

Operands:

- AdcMode – Operand activates the ADC mode on the selected current measurement block

Operand label	Operand description
off	The current measurement block compares the current flowing in the actuator with a threshold (nominal behavior).
on	The current measurement block performs an analog-to-digital conversion of the current flowing in the actuator

- DacTarget – Operand defines the current measurement block DAC to be set in ADC mode

Operand label	Operand description
sssc	DAC of the same microcore same channel
ossc	DAC of the other microcore same channel
ssoc	DAC of the same microcore other channel
osoc	DAC of the other microcore other channel

Table 98. stal instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	1	0	1	AdcMode	DacTarget	

Source code example:

```
*### Software AtoD Conversion routine in Channel 1 Microcore 0 ####
#define Tadc_sp 0;                          * Define Sampling time ADC (11ck_ofscmp) to be stored in Data RAM address 0
#define ADC_results 1;                      * Define ADC results to be stored in Data RAM address 1

ADCinit:    cwer sample tc2 row1;           * Create wait table entry 1 when tc2 is reached go to Sample label (22us recommended)

Convert:    ldcd rst _ofs keep keep Tadc_sp c2; * Load the length of the sampling time in counter 2
            stadc on sssc;                  * Set current sense1 in adc mode and start acquisition
            wait row1;                      * Wait until the sample time is done (refer to ADC init)

Sample:     store dac_sssc ADC_results;     * Copy adc results inside

ADCdisable: stadc off sssc;                 * Stop the ADC mode (note that if another conversion is needed 2 ck_ofscmp clk cycles are needed
                                            between stadc on and stadc off instruction)
```

stal

Description:

Sets the arithmetic logic mode. This mode is the set according to the bits A1 and A0 of the ALU condition register (arith_reg). This instruction configures the behavior of addition and subtraction instructions only. All other math instructions (multiply, shift, bitwise) are not affected by this instruction. The addition and subtraction results are affected only if one of the 'saturation' modes is selected.

With 'saturation' enabled the results is bounded by the natural limits of the 16-bit register (max signed = 0x7FFF) ALU operations behavior is affected by the arithmetic logic mode ModeAL as described by the following:

The ALU instruction operands are handled as a C-complement number (signed number). If the resulting value exceeds the result register capacity, it leads to overflow detection but no saturation.

The ALU instruction operands are handled as a C-complement number (signed number). If the resulting value exceeds the result register capacity, it leads to overflow detection and saturation.

The ALU instruction operands are handled as a positive number (unsigned number). If the resulting value exceeds the result register capacity it leads to overflow detection but no saturation.

The ALU instruction operands are handled as a positive number (unsigned number). If the resulting value exceeds the result register capacity it leads to overflow detection and saturation.

The ModeAL reset value is al3.

Assembler syntax: `stal ModeAL;`

Operands:

- ModeAL – Operand defines the ALU behavior selected

Operand label	Operand description
al1	Two's complement number without overflow saturation
al2	Two's complement number with overflow saturation
al3	Positive number without overflow saturation
al4	Positive number with overflow saturation

Table 99. stal instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	1	1	1	1	ModeAL	

Source code example:

```
*##### Set saturation mode al2 to limit max/min values #####
Sat_test:       stal al2;                   * Set saturation mode to al2
                ldirh 7Fh rst;              * Set 7Fh in the ir MSB and reset ir LSB
                ldirl FDh _rst;             * Set FFh in the ir LSB and keep the MSB -> ir = 7FFDh
                addi ir 15 ir;              * Add 15d to ir -> ir = ir +15d = 7FFDh + 15d = 800C but since saturation is enabled ir is equal to max
FFFFh
```

stcrb

Description:

Sets the logic level value individually with the Logic operand of each selected bit CrbSel of the control register.

Assembler syntax: `stcrb Logic CrbSel;`

Operands:

- Logic – Operand defines the logic level value

Operand label	Operand description
low	Low level
high	High level

- CrbSel – Operand defines the control register bit to be selected

Operand label	Operand description
b8	Control register bit 8
b9	Control register bit 9
b10	Control register bit 10
b11	Control register bit 11
b12	Control register bit 12
b13	Control register bit 13
b14	Control register bit 14
b15	Control register bit 15 (MSB)

Table 100. stcrb instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	1	0	1	0	Logic	0	CrbSel		

stcrt

Description:

Each microcore shares the ch_rxtx register with the other microcores can read the shared register of another microcore. This instruction selects the microcore's shared register accessed by the microcore executing the stcrt instruction.

The UcId reset value is sssc.

Assembler syntax: `stcrt UcId;`

Operands:

- UcId – Operand defines the microcore shared register to be access.

Operand label	Operand description
sssc	The microcore executing the code
ossc	The other microcore in the same channel
ssoc	The same microcore in the other channel
osoc	The other microcore in the other channel
sumh	The sum of bits H
suml	The sum of bits L

Table 101. stcrt instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	0	1	1	UcId	

Source code example:

```
*##### Exchange data between Channel 1 uCore 0 and Channel 2 uCore1 ######

*##### Channel 1 uCore0 add 40h in the rxtx register ####
Ld_rxtx:        ldirl 40h rst;              * Load ir LSB register with 40h and reset ir MSB
                cp ir rxtx;                 * Channel1 Ucore0 rxtx register loaded with 40h
......
*##### End of Channel1 uCore0 ######


*##### Channel 2 uCore1 read ch1uc0_rxtx register and save it in ir register ####
Access_rxtx:    stcrt osoc;                 * Access set to other microcore previous channel, in this case Channel1 uCore0
                cp rxtx ir;                 * Copy rxtx coming from ch1_uc0 -> ir = 40h
.......
*##### End of Channel2 uCore1 ######
```

stdcctl

Description:

Selects if the DCDC must be controlled by the microcore (sync) or perform the automatic current regulation (async) between threshold 4l and 4h.

The ModeDC reset value is sync.

Assembler syntax: `stdcctl ModeDC;`

Operands:

- ModeDC – Operand defines the DC-DC control mode

Operand label	Operand description
sync	DCDC is controlled by the microcore
async	DCDC performs an automatic current control between threshold 4l and 4h

Table 102. stdcctl instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0	0	0	1	ModeDC

Source code example:

```
DCDC_set:   dfsct undef ls7 undef;          * Set lowside 7 as shortcut 2 needed for stdcctl instruction
            ......                          * Need to configure dac settings and wait table
                                            (See Including a data RAM address definition file on page 135)
            stdcctl async;                  * Enable DCDC resonant mode
```

stdm

Description:

The DAC registers address (dac_sssc, dac_ossc, dac_ssoc, dac_osoc) in the internal data memory map are split in two slices (dac value, offset register):

This instruction selects which slice(s) is accessed.

The dac4h4n_boost address in the internal data memory map can refer to three registers (dac4h value, dac4neg value, dac boost value); this same instruction select which of the three registers is accessed.

The dac_osoc_batt address in the internal data memory map can refer to two registers (dac_osoc, adc batt result); this same instruction select which of the two register is accessed.

dac_access_mode/dac4h_access_mode: the dac value (for the dac address) or the dac4H value (for the dac4h4n_boost address) is accessed. The result is available in the 8 lower bits.

offset_access_mode/dac4neg_access_mode: the offset register (for the dac address) or the dac4neg value (for the dac4h4n_boost address) is accessed. The result is available in the 13-8 bits if reading an offset, in the 11-8 bits if reading dac4neg.

full_access_mode/dac4h4n_access_mode: both the dac value and the offset register (for the dac address) or both the dac4h and the dac4n value (for the dac4h4n_boost address) is accessed.

adc_batt_access_mode/dac_boost_access_mode: nothing (for the first three dac address), the vbatt measurement adc result for the dac_osoc_batt address or the value of the dac boost (for the dac4h4n_boost address) is accessed.

The ModeDAC reset value is dac.

Assembler syntax: `stdm ModeDAC;`

Operands:

- ModeDAC – Operand defines the DAC access mode

Operand label	Operand description
null	adc_batt_access_mode/dac_boost_access_mode
dac	dac_access_mode/dac4h_access_mode
offset	offset_access_mode/dac4n_access_mode
full	full_access_mode/dac4h4n_access_mode

Table 103. stdm instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	0	0	1	ModeDAC	

stdrm

Description:

Sets the Data RAM read mode.

The possible read modes according to the ModeDRM operand are:

dram_word_mode: all 16 bits are accessed

dram_lowbyte _mode: only the 8 LSBs of the source Data RAM are accessed. The result is available in the 8 lower bits of the destination register. The upper 8 bits of the destination register is set to 00h.

dram_highbyte_mode: only the 8 MSBs of the source Data RAM are accessed. The result is available in the 8 lower bits of the destination register. The upper 8 bits of the destination register is set to 00h.

dram_swapbyte_mode: the 8 LSBs and 8 MSBs of the source dram are accessed swapped and is available at the destination register. This read mode is valid after the load and ldcd instructions following this stdrm instruction.

The ModeDRM reset value is word.

Assembler syntax: `stdrm ModeDRM;`

Operands:

- ModeDRM – Operand defines the Data RAM read access

Operand label	Operand description	Operand binary value
word	dram_word_mode	00
low	dram_lowbyte _mode:	01
high	dram_highbyte_mode	10
swap	dram_swapbyte_mode	11

Table 104. stdrm instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	0	0	0	ModeDRM	

steoa

Description:

Enables or disables the end-of-actuation mode for all the high-side predrivers the microcore has enabled to drive by means of the Switch operand.

The V_{SRC} threshold monitoring of the related predrivers can be disabled by setting the operand Mask. The Mask default value is nomask.

The Switch default value is bsneutral.

Assembler syntax: `steoa Mask Switch;`

Operands:

- Mask – Operand sets the V_{DS} threshold mask

Operand label	Operand description
nomask	V_{SRC} threshold monitoring of the selected HS is unchanged
mask	V_{SRC} threshold monitoring of the selected HS is masked to zero

- Switch – Operand sets the end-of-actuation mode

Operand label	Operand description
keep	Maintain the previous values
bsoff	Bootstrap switch is forced off
bson	Bootstrap switch can be enabled even if no low-side predriver is switched on
bsneutral	Bootstrap control is not affected

Table 105. steoa instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	0	1	0	0	Mask	Switch	

stf

Description:

Sets the logic level value with the Boolean Logic of the selected flag. The flag is selected according the FlgSel operand.

Assembler syntax: `stf Logic FlgSel;`

Operands:

- Logic – Operand defines the logic level value

Operand label	Operand description
low	Low level
high	High level

- FlgSel – Operand defines the flag bit to be selected

Operand label	Operand description
b0	Flag bit 0
b1	Flag bit 1
b2	Flag bit 2
b3	Flag bit 3
b4	Flag bit 4
b5	Flag bit 5
b6	Flag bit 6
b7	Flag bit 7
b8	Flag bit 8
b9	Flag bit 9
b10	Flag bit 10
b11	Flag bit 11
b12	Flag bit 12
b13	Flag bit 13
b14	Flag bit 14
b15	Flag bit 15

Table 106. stf instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	1	1	1	Logic	FlgSel			

stfw

Description:

Defines the freewheeling output modes. Freewheeling control is automatic or manual according to the FwMode operand.

For each HS there are two possible freewheeling outputs. The output is selected in the fw_link (refer to Fw_link (169h)) register. The FwMode operand is a Boolean defining the control mode:

if Shortcut1 is HS1, then LS5 is set as freewheeling predriver

if Shortcut1 is HS2, then LS6 is set as freewheeling predriver

if Shortcut1 is HS3, then LS7 is set as freewheeling predriver

if Shortcut1 is HS4, then HS5 is set as freewheeling predriver

if Shortcut1 is HS5, then LS4 is set as freewheeling predriver.

The shortcuts are set using the dfsct instruction.

This operation is successful only if the microcore has the right to drive the output it has defined as shortcut1 (output access register).

The FwMode reset value is manual.

Assembler syntax: `stfw FwMode;`

Operands:

- FwMode – Operand defines the freewheeling mode

Operand label	Operand description
manual	Freewheeling manual control
auto	Freewheeling automatic control

Table 107. stfw instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	0	0	0	FwMode

Source code example:

```
*### Freewheeling between HS1 and LS1 (fw_link register needs to be set 169h) ####

Short_def:    dfsct hs1 hs2 ls2;          * Shortcut 1 is the one used for freewheeling so hs1 needs to be set in first position since the FW is LS1

PeakON:       stos on off on;             * During Boost phase turn ON hs1 hs2 ls2 and keep ls1 OFF
              stfw auto;                  * Start the automatic freewheeling (has to be done after high side turn ON to avoid unwanted turn ON
of freewheeling low side after the stfw auto instruction)
              ......
PeakOFF:      stos off off on;            I* hs1 off, hs2 off, ls2 on and ls1 ON because of auto freewheeling
              ......
InjOFF:       stos off off off;           I* hs1 off, hs2 off, ls2 off and ls1 ON because of auto freewheeling
              stfw manual;                I* all high side and low side are OFF and freewheeling is now disabled
```

stgn

Description:

Sets the gain of an operational amplifier with the Gain operand used to measure the current flowing through the actuator sense resistor. The operational amplifier is selected according to the OpAmp operand.

The operation is successful only if the microcore has the right to access the related current measurement block. The access right is granted by setting the related bits in the Cur_block_access_part1 register (188h), and Cur_block_access_part2 Register (189h).

The other channel is selected by SPI register bit (refer to Dac_rxtx_cr_config (112h, 132h))

The Gain reset value is gain 5.8.

Assembler syntax: `stgn Gain OpAmp;`

Operands:

- Gain – Operand defines the current measure operational amplifier gain

Operand label	Operand description	Operand binary value
gain5.8	Operational amplifier gain set to 5.8	00
gain8.7	Operational amplifier gain set to 8.7	01
gain12.6	Operational amplifier gain set to 12.5	10
gain19.3	Operational amplifier gain set to 19.3	11

- OpAmp – Operand defines the current measure operational amplifier gain to be set

Operand label	Operand description	Operand binary value
sssc	Current measure operational amplifier of the same microcore same channel	00
ossc	Current measure operational amplifier of the other microcore same channel	01
ssoc	Current measure operational amplifier of the same microcore other channel	10
osoc	Current measure operational amplifier of the other microcore other channel	11

Table 108. stgn instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	0	1	1	Gain		OpAmp	

Source code example:

```
*#### Set gain of Current Sense 1 to 12.6 using Channel 1 Microcore 0 ######
init_gain:    stgn gain12.6 sssc;              * Set the gain of the opamp of the current measure block 1 (same microcore same channel)
```

stirq

Description:

Set the IRQB output pin

The Logic reset value is high.

Assembler syntax: `stirq Logic;`

Operands:

- Logic – Operand defines the logic level of the IRQB pin

Operand label	Operand description
low	Low level
high	High level

Table 109. stirq instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	0	0	1	Logic

sto

Description:

Sets the state with the Out operand for the selected output according to the OutSel operand.

The operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (160h, 161h, 162h, 163h, 164h, 165h) configuration registers.

Assembler syntax: `sto OutSel Out;`

Operands:

- OutSel – Operand defines the handled output

Operand label	Operand description
hs1	High-side predriver 1
hs2	High-side predriver 2
hs3	High-side predriver 3
hs4	High-side predriver 4
hs5	High-side predriver 5
ls1	Low-side predriver 1
ls2	Low-side predriver 2
ls3	Low-side predriver 3
ls4	Low-side predriver 4
ls5	Low-side predriver 5
ls6	Low-side predriver 6
ls7	Low-side predriver 7
undef	Undefined

- Out – Operand sets output state

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Output disabled
on	Output enabled
toggle	Reverse the previous setting

Table 110. sto instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	1	0	OutSel				Out	

stoc

Description:

Enables or disables the offset compensation with the operand Ctrl on the current measurement block specified according to the DacTarget operand.

The operation is successful only if the microcore has the right to access the related current measurement block. The access right is granted by setting the related bits in the Cur_block_access_part1 register (188h), and Cur_block_access_2 Register (189h).

The other channel is selected by SPI register bit (refer to Dac_rxtx_cr_config (112h, 132h)).

The Ctrl reset value is off for all current measurement blocks.

Assembler syntax: `stoc Ctrl DacTraget;`

Operands:

- Ctrl – Operands sets offset compensation state

Operand label	Operand description
off	Disable the offset compensation
on	Enable the offset compensation

- DacTarget – Operand defines the current measurement block

Operand label	Operand description
sssc	DAC of the same microcore same channel
ossc	DAC of the other microcore same channel
ssoc	DAC of the same microcore other channel
osoc	DAC of the other microcore other channel

Table 111. stoc instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	1	0	Ctrl	DacTarget	

Source code example:

```
*### Software AtoD Conversion routine in Channel 1 Microcore 0 ####
#define Toffset_comp 0;                    * The duration for the offset compensation is worst case 31 x ck_ofscmp, it is define in DRAM
                                           address 2

Offcmpinit:     cwer OffcmpOff tc2 row1;   * Create wait table entry 1 when tc2 is reached go to OffCmpOff label

Offcmp:         ldcd rst _ofs keep keep Toffset_cmp c1; * Load the length of the sampling time in counter 2
                stoc on sssc;              * Set offset compensation ON for current sense 1 (no current has to flow in the sense)
                wait row12;                * Wait until the offset compensation is done or row 5 event (i.e.: start event) occurred

OffcmpOff:      stoc off sssc;             * Turn OFF offset compensation
```

store

Description:

Copies the content of the op1 source register in a Data RAM line defined by the 6-bit Data RAM address Dram.

The operand Dram can be identified with a univocal label. The compiler automatically substitutes the 'define' label (if used) with the suitable Data RAM address.

The Data RAM address is accessed according to the Boolean operand Ofs using the: Immediate addressing mode (IM).

Indexed addressing mode (XM). In that case, the address base is added to the address Dram. The address base is set using the stab instructions.

Assembler syntax: `store op1 Dram Ofs;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.3 "UcReg subset"
- Dram – Operand defines the 6-bit DRAM address
- Ofs – Operands sets data RAM addressing mode

Operand label	Operand description
_ofs	Data RAM immediate addressing mode (IM)
ofs	Data RAM indexed addressing mode (XM)

Table 112. store instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	Ofs	op1					Dram						0	1

Source code example:

```
*### Store a Data in DRAM address 0####
#define Testresults 0;                    * Define Testresults in the Data RAM address 0

Test:           ldirl 45 rst;             * Load 45 in ir register and reset the MSB -> ir = 45d
                addi ir 9 r1;             * Add ir + 9 and save results in r1 -> 45 + 9 = 54 in r1

Store_resutls:  store r1 Testresults _ofs; * Store results inside dataram address 0 called Testresults, no offset required (no stab instruction)
```

stos

Description:

Sets the state of three outputs Out1, Out2 and Out3 previously defined as shortcuts with the dfsct instruction.

The operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

Assembler syntax: `stos Out1 Out2 Out3;`

Operands:

- Out1, Out2, and Out3 – Operands sets output state

Operand label	Operand description
keep	No changes, maintains the previous setting
off	Output disabled
on	Output enabled
toggle	Reverse the previous setting

Table 113. stos instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	0	1	Out1		Out2		Out3	

stslew

Description:

Defines the outputs slew rate mode with the Boolean SlMode.

The operation is successful only if the microcore has the right to drive the related outputs. The drive right is granted by setting the related bits in the Out_acc_ucX_chY (184h, 185h, 186h, 187h) configuration registers.

The SlMode reset value is normal.

When switching the slew-rate from slow to fast, the new slew-rate is valid after typically one ck cycle (166 ns considering fCK = 6.0 MHz). When switching from fast to slow, it takes typically four ck cycles (666 ns considering fCK = 6.0 MHz) until the new slew-rate is effective.

Assembler syntax: `stslew SlMode;`

Operands:

- SlMode – Operands sets outputs slew rate mode

Operand label	Operand description
normal	The outputs slew rate is set by an SPI register
fast	The outputs slew rate is the highest one

Table 114. stslew instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	0	1	1	SlMode

stmfm (PT2001MAE only)

Description:

The stmfm instruction provides a way to switch the S_HSx and D_LSx multiplexer to the predriver, which is currently set as a shortcut for the microcore calling the instruction.

In addition to this switching capability, it is possible to enable or disable the analog OAx output of the device

Assembler syntax: `stmfm d_ls s_hs oax Switch;`

Operands:

- d_ls – Operand select the D_LSx multiplexer input
- s_hs – Operand select the S_HSx multiplexer input
- oax – Operand select the OAx used for the measurement function
- Switch – Operand enable/disable the OAx (oa_enable)

Table 115. d_ls

Operand label	Operand description	Operand binary value
sh2	D_LS of shortcut 2 selected	00
sh3	D_LS of shortcut 3 selected	01
agnd	AGND selected	10
keep	Setup not changed	11

Table 116. s_hs

Operand label	Operand description	Operand binary value
sh1	S_HS of shortcut 1 selected	00
sh2	S_HS of shortcut 2 selected	01
agnd	AGND selected	10
keep	Setup not changed	11

Table 117. oax

Operand label	Operand description	Operand binary value
oa1	OA_1 selected	00
oa2	OA_2 selected	01
non	No OAx are selected (they can be used for current sense)	11

Table 118. Switch

Operand label	Operand description	Operand binary value
en	oax enabled	0
dis	oax disabled	1

Instruction format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	Switch	oax		s_hs		d_ls		1	0	0	0

stsrb

Description:

Sets individually the logic level value with the Logic operand of each selected bit SrbSel of the status register (status_reg_uc0 (105h, 125h) and status_reg_uc1 (106h, 126h)).

Assembler syntax: `stsrb Logic SrbSel;`

Operands:

- Logic – Operand defines the logic level value

Operand label	Operand description
low	Low level
high	High level

- SrbSel – Operand defines the status register bit to be selected

Operand label	Operand description
b0	Status register bit 0 (LSB)
b1	Status register bit 1
b2	Status register bit 2
b3	Status register bit 3
b4	Status register bit 4
b5	Status register bit 5
b6	Status register bit 6
b7	Status register bit 7
b8	Status register bit 8
b9	Status register bit 9
b10	Status register bit 10
b11	Status register bit 11
b12	Status register bit 12
b13	Status register bit 13
b14	Status register bit 14
b15	Status register bit 15 (MSB)

Table 119. stsrb instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	1	0	0	0	Logic	SrbSel			

sub

Description:

Subtracts the value contained in the op2 register from the value contained in op1 register and places the result in the res register.

Operation: (Source1) – (Source2) → (Destination)

Assembler syntax: `sub op1 op2 res;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- op2 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- res – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- RZ – Addition or subtraction result is zero
- RS – Addition or subtraction result is negative
- UU – Unsigned underflow
- UO – Unsigned overflow
- SU – Signed underflow
- SO – Signed overflow

Table 120. sub instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	res			1	op2			op1		

subi

Description:

Subtracts the value contained in the Imm register from the value contained in the op1 register and places the result in the res register.

Operation: (Source1) – Immediate value → (Destination)

Assembler syntax: `subi op1 Imm res;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Imm – The Imm 4-bit immediate data register
- res – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Condition register:

- RZ – Addition or subtraction result is zero
- RS – Addition or subtraction result is negative
- UU – Unsigned underflow
- UO – Unsigned overflow
- SU – Signed underflow
- SO – Signed overflow

Table 121. subi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	res			Imm				op1		

swap

Description:

Swaps the high byte and the low byte of the register op1.

Operation: (Source)[0:7] < → (Source)[8:15]

Assembler syntax: `swap op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Table 122. swap instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	1	0	0	op1		

swi

Description:

Enables or disables all software interrupts and from start edges for a microcore. HW interrupts from automatic diagnosis, driver disable or loss of clock are not disabled.

The Switch reset value is on (all SW interrupts enabled).

Assembler syntax: `swi Switch;`

Operands:

- Switch – Operands enable or disable SW interrupts

Operand label	Operand description
on	SW interrupts on
off	SW interrupts off

Table 123. swi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	1	0	0	1	0	1	0	Switch

toc2

Description:

Converts the integer value contained in the AluReg register into a two's complement format.

If the conversion bit in the arithmetic condition register is zero, the 'toc2' instruction makes the most significant bit in the operand register zero.

If the conversion bit is one, then it returns the two's complement of the operand (bits[14:0] only) register.

Assembler syntax: `toc2 op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"

Table 124. toc2 instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	1	0	0	1	1	0	op1		

toint

Description:

Convert the two's complement value contained in op1 register to integer format.

The toint instruction retains the original value in the operand register op1 when its MSB bit is zero. If the MSB is 1, then it returns the two's complement of the operand register (op1[14:0]).

The toint instruction also saves the MSB of the operand op1 in the conversion bit CS of the arithmetic condition register arith_reg.

The MSB of the operand is either XORed with the existing conversion bit CS of the ALU condition register (if the instruction is called with the _rst parameter) or replaces it (if the instruction is called with the rst parameter).

Assembler syntax: `toint op1 Rst;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Rst – Operand defines if the conversion bit CS of the ALU condition register is reset

Operand label	Operand description
_rst	The existing conversion bit CS is XORed with the op1 MSB
rst	The existing conversion bit CS is set according to the op1 MSB

Condition register:

- CS – Last conversion sign

Table 125. toint instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	1	0	0	1	0	Rst	op1		

wait

Description:

Stops the program counter (uPC) incrementing and waits until at least one of the enabled wait conditions is satisfied. When one of the conditions is satisfied, the program counter is moved to the corresponding destination.

The possible wait conditions, along with the corresponding destinations, are stored in the wait table by means of the cwer and cwef instructions.

The active wait table rows are enabled according to the WaitMask 6-bit operand.

Assembler syntax: `wait WaitMask;`

Operands:

- WaitMask – Operand defines the active wait table rows

Operand label	Operand description	Operand binary value
always	No wait table row enabled, infinite loop	000000
row1	Wait table row 1 enabled	000001
row2	Wait table row 2 enabled	000010
row12	Wait table row 1,2 enabled	000011
row3	Wait table row 3 enabled	000100
row13	Wait table row 1,3 enabled	000101
row23	Wait table row 2,3 enabled	000110
row123	Wait table row 1,2,3 enabled	000111
row4	Wait table row 4 enabled	001000
row14	Wait table row 1,4 enabled	001001
row24	Wait table row 2,4 enabled	001010
row124	Wait table row 1,2,4 enabled	001011
row34	Wait table row 3,4 enabled	001100
row134	Wait table row 1,3,4 enabled	001101
row234	Wait table row 2,3,4 enabled	001110
row1234	Wait table row 1,2,3,4 enabled	001111
row5	Wait table row 5 enabled	010000
row15	Wait table row 1,5 enabled	010001
row25	Wait table row 2,5 enabled	010010
row125	Wait table row 1,2,5 enabled	010011
row35	Wait table row 3,5 enabled	010100
row135	Wait table row 1,3,5 enabled	010101
row235	Wait table row 2,3,5 enabled	010110
row1235	Wait table row 1,2,3,5 enabled	010111
row45	Wait table row 4,5 enabled	011000
row145	Wait table row 1,4,5 enabled	011001

Operand label	Operand description	Operand binary value
row245	Wait table row 2,4,5 enabled	011010
row1245	Wait table row 1,2,4,5 enabled	011011
row345	Wait table row 3,4,5 enabled	011100
row1345	Wait table row 1,3,4,5 enabled	011101
row2345	Wait table row 2,3,4,5 enabled	011110
row12345	Wait table row 1,2,3,4,5 enabled	011111
row6	Wait table row 6 enabled	100000
row16	Wait table row 1,6 enabled	100001
row26	Wait table row 2,6 enabled	100010
row36	Wait table row 3,6 enabled	100011
row46	Wait table row 4,6 enabled	100100
row56	Wait table row 5,6 enabled	100101
row126	Wait table row 1,2,6enabled	100110
row136	Wait table row 1,3,6 enabled	100111
row146	Wait table row 1,4,6 enabled	101000
row156	Wait table row 1,5,6 enabled	101001
row236	Wait table row 2,3,6 enabled	101010
row246	Wait table row 2,4,6 enabled	101011
row256	Wait table row 2,5,6 enabled	101100
row346	Wait table row 3,4,6 enabled	101101
row456	Wait table row 4,5,6 enabled	101110
row1236	Wait table row 1,2,3,6 enabled	101111
row1246	Wait table row 1,2,4,6 enabled	110000
row1256	Wait table row 1,2,5,6 enabled	110001
row1346	Wait table row 1,3,4,6 enabled	110010
row1356	Wait table row 1,3,5,6 enabled	110011
row1456	Wait table row 1,4,5,6 enabled	110100
row2346	Wait table row 2,3,4,6 enabled	110101
row2356	Wait table row 2,3,5,6 enabled	110110
row2456	Wait table row 2,4,5,6 enabled	110111
row3456	Wait table row 3,4,5,6 enabled	111000
row12346	Wait table row 1,2,3,4,6 enabled	111001
row12356	Wait table row 1,2,3,5,6 enabled	111010
row12456	Wait table row 1,2,4,5,6 enabled	111011
row13456	Wait table row 1,3,4,5,6 enabled	111100
row123456	Wait table row 1,2,3,4,5,6 enabled	111101

Table 126. wait instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	0	0	WaitMask (row 6 disabled)				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	0	1	1	WaitMask (row 6 enabled)				

wrspi

Description:

Requests an SPI backdoor write.

The address must previously be defined in the SPI address register spi_add.

The data must previously be defined in the SPI data register spi_data.

The wrspi instruction requires 2 ck cycles to complete. The SPI address register and SPI data register must not be changed on the following instruction, otherwise the operation fails and the written data is dummy.

Assembler syntax: `wrspi;`

Table 127. wrspi instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0	0	0	0	1

Source code example:

```
*#### This code writes the value 125h inside the register at address 171h ###

SPI_Init:   slsa ir;              * Configure SPI accesses to use 'ir' for addresses
Ld_Data:    ldirh 01h _rst;       * Load 01h in the ir MSBregister
            ldirl 25h _rst;       * Load 25h in the ir LSB register -> ir = 125h
            cp ir spi_data;       * Load 125h into spi_data
Ld_Add:     ldirh 01h _rst;       * Load 01h in the ir MSB register (useless in this case since already loaded before)
            ldirl 71h _rst;       * Load 71h in the ir LSB register -> ir = 171h
SPI_Write:  wrspi;                * Write SPI using ir for address (171h) and spi_data for data (125h)
```

xor

Description:

Applies the XOR-mask contained in the Ir register to the value contained in the op1 register and places the result in the op1 register. The initial data stored in the op1 register is lost.

Operation: (Source) XOR Immediate register → (Source)

Assembler syntax: `xor op1;`

Operands:

- op1 – One of the registers listed in the operand Section 3.1.1 "AluReg subset"
- Ir – The ALU immediate register

Condition register:

- MN – Mask result is 0000h
- MM – Mask result is FFFFh

Table 128. xor instruction format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	0	1	1	0	1	0	op1		

4 Specific assembler language

The PT2001 requires microcode to enable most of its functions. The main benefit is the large flexibility in configuring the device. This microcode is defined by the software engineer in a source file, coding the PT2001 specific instructions in assembler language.

The extension *.dfi or *.psc is generally used for the source file. Any other extension can be used as a source code extension, with the exception of assembler's input file extensions (*.link, *.xml, *.key) or output file extensions. (*.cip, *.hex, *.bin, *.asm, *.log, *.reg, *.cip.bin, *.cip.hex).

The assembler language coding rules are defined in the following sections.

4.1 Writing an instruction

Instructions allow the software designer to define the behavior that is executed independently by each microcore.

The allowed instructions and parameters are only the ones defined in the default instructions library (syntax.xml) or the custom syntax files. All the instructions must be followed by the mandatory parameters. All the instructions must terminate with the character ';'. The instruction syntax is as follows:

InstructionName Parameter1NameOrValue Parameter2NameOrValue...;

The instruction and parameter descriptions are provided in Section 3 "Instruction set and subsets". The instructions and the associated parameters are case sensitive. They can only be placed:

- At the beginning of the line
- Or after the end-of-comment field character '*'
- Or after a valid Label

One instruction per source file line or per include line is allowed. Below is an example:

stf low b0;

4.2 Inserting a comment field

The source code file supports the addition of comments. The comment fields are identified with:

- Two '*' characters, one placed before and the other after the comment text
- One '*' symbol placed before comment text. In this case, all characters up to the end of the line are considered as part of the comment.

The comment field syntax is as follows:

Comment

**Comment*

Below are some examples:

Put the channel in error state SWInterruptRoutine: stflowErrorFlag;

SWInterruptRoutine: stflowErrorFlag; *Put the channel in error state

4.3 Defining a constant

The software designer can use a constant value label, instead of using a number as an instruction parameter (define). This constant definition helps to make the source code more readable. The define function is used for a constant value definition that is used locally in the source code. The constant definitions must terminate in the character ';'. The define syntax is as follows:

#define *SymbolName SymbolValue*;

The constant definitions are placed:

- At the beginning of the line
- Or after the final comment field character '*'

The constant definition must be placed in an instruction line. No other item, such as an instruction, label or include statement, is allowed in a constant definition line. All define statements must be unique, that is not already used as the *SymbolName* of a line label, and cannot have the same name as an instruction or a parameter. The define name (*SymbolName*) cannot start with a number but can contain one or several numbers. It must not include spaces. The Define arguments *SymbolName* and *SymbolValue* are mandatory. See the example below:

#define ErrorFlag 10;

4.4 Including a data RAM address definition file

The assembler has the ability to manage a nested file structure. The sub files called in the main source file are known as definition files. These definition files are commonly used for variable definition dedicated to device Data RAM. However, any instruction or label can be used in definition files.

All the include statements must terminate with the character ';'. A valid file name must be placed between two apostrophe characters ('*text*').

The include declaration must be placed:

- At the beginning of the line
- Or after the end comment character '*' The include syntax is as follows:

#include '*Filename*.def';

Use of nested *include* is not permitted.

Note that *.def or any other extension can be used as a definition file extension, with the exception of assembler's input file extensions (*.dfi, *.link, *.xml) or output file extensions. (*.cip, *.hex, *.bin, *.asm, *.log, *.reg, *.cip.bin, *.cip.hex).

See the example below:

#include '*Source1.def*'; *include variable to the source.dfi defined in the definition file

4.5 Using a line label

The assembler can manage line labels. This kind of label is used to replace a line number called as an instruction parameter. The assembler immediately replaces the label with the corresponding line number at the time of the source code assembly.

The label syntax is as follows:

LabelName:

The label refers to a line code where the label is set. For example, if the label *Init* is located on line 7 any instruction using this label refers to line 7.

Labels must be placed at the beginning of the line or after the final character of a comment field. These labels end with the symbol ':' . All labels must be followed by an instruction. They must be unique, must not already have been used as a *SymbolName* in a Define statement, and cannot be an instruction or parameter name. The *LabelName* can contain numbers, but cannot start with a number and must not include spaces. An example follows:

SWInterruptRoutine:

4.6 Numbering convention

A parameter can either be a parameter name associated with a value defined in the syntax file or it can be a numeric value. When a numeric value is used, the parameter is decimal. Three formats are possible:

- By default, the value is decimal (no suffix)
- A 'h' specifies that the value is hexadecimal
- A 'b' suffix specifies that the value is binary An example follows:

ldirl 10 _rst; *number 10

ldirl 10h _rst; *number 16

ldirl 10b _rst; *number 2

4.7 Conditional assembly

The assembler includes a basic `IF` function (conditional assembly). This function is 'static' so the `IF` function branch is considered at the time of assembly. The `IF` function syntax is as follows:

```
#IF Condition
Instructions1
#ELSEIF
Instructions2
#ENDIF
```

The conditional assembly considers a branch only if its parameter is defined (whatever its value may be). Using an `ELSE` branch is optional. All the instructions placed before the `#IF` label and after the `#ENDIF` label are excluded from the conditional code block and are assembled. Only one level of condition assembly is supported, so an `IF` function cannot be nested within another `IF` function. Consider the example below:

```
 #define DDI 1; *define optional in this case.
 #IF DDI
 jmpr DDI_Init; *DDI constant is defined so this condition is met
 *In this case the program counter jumps to the DDI_Init label line
 #ELSE
 jmpr GDI_Init; *GDI constant is not defined so this condition is never met
#ENDIF
```

5 Example source code

This code can be used with the FRDMPT2001EVM.

5.1 Channel 1 - Ucore0 - controls injectors 1 and 2

```
* ### Channel 1 - uCore0 controls the injectors 1 and 2 ###

* ### Variables declaration  ###

* Note: The data are stored into the dataRAM of the channel 1.
#define Iboost 0;                          * The boost phase current value is stored in the Data RAM address 0
#define Ipeak 1;                           * The peak phase current value is stored in the Data RAM address 1
#define Ihold 2;                           * The hold phase current value is stored in the Data RAM address 2
#define Tpeak_off 3;                       * The peak off phase time is stored in the Data RAM address 3
#define Tpeak_tot 4;                       * The peak phase duration is stored in the Data RAM address 4
#define Tbypass 5;                         * The bypass phase time is stored in the Data RAM address 5
#define Thold_off 6;                       * The peak phase duration is stored in the Data RAM address 6
#define Thold_tot 7;                       * The peak phase duration is stored in the Data RAM address 7
* Note: The Thold_tot variable defines the current profile time out. The active STARTx pin is expected to toggle in is low state before this time out.

* ### Initialization phase ###
init0:      stgn gain12.6 sssc;            * Set the gain of the opamp of the current measure block 1
            ldjr1 eoinj0;                  * Load the eoinj line label Code RAM address into the register jr1
            ldjr2 idle0;                   * Load the idle line label Code RAM address into the register jr2
            cwef jr1 _start row1;          * If the start signal goes low, go to eoinj phase

* ### Idle phase- the uPC loops here until start signal is present ###
idle0:      joslr inj1_start start1;       * Perform an actuation on inj1 if start 1 (only) is active
            joslr inj2_start start2;       * Perform an actuation on inj2 if start 2 (only) is active
            jmpf jr1;                      * If more than 1 start active at the same time(or none), no actuation

* ### Shortcuts definition per the injector to be actuated ###
inj1_start: dfsct hs1 hs2 ls1;             * Set the 3 shortcuts: VBAT, VBOOST, LS
            jmpr boost0;                   * Jump to launch phase

inj2_start: dfsct hs1 hs2 ls2;             * Set the 3 shortcuts: VBAT, VBOOST, LS
            jmpr boost0;                   * Jump to launch phase

* ### Launch phase enable boost ###
boost0:     load Iboost dac_sssc _ofs;     * Load the boost phase current threshold in the current DAC
            cwer peak0 ocur row2;          * Jump to peak phase when current is over threshold
            stf low b0;                    * set flag0 low to force the DC-DC converter in idle mode
            stos off on on;                * Turn VBAT off, BOOST on, LS on
            wait row12;                    * Wait for one of the previously defined conditions

* ### Peak phase continue on Vbat ###
peak0:      ldcd rst _ofs keep keep Tpeak_tot c1; * Load the length of the total peak phase in counter 1
            load Ipeak dac_sssc _ofs;      * Load the peak current threshold in the current DAC
            cwer bypass0 tc1 row2;         * Jump to bypass phase when tc1 reaches end of count
            cwer peak_on0 tc2 row3;        * Jump to peak_on when tc2 reaches end of count
            cwer peak_off0 ocur row4;      * Jump to peak_off when current is over threshold
            stf high b0;                   * set flag0 high to release the DC-DC converter idle mode

peak_on0: stos on off on;                  * Turn VBAT on, BOOST off, LS on
            wait row124;                   * Wait for one of the previously defined conditions

peak_off0: ldcd rst ofs keep keep Tpeak_off c2; * Load in the counter 2 the length of the peak_off phase
            stos off off on;               * Turn VBAT off, BOOST off, LS on
            wait row123;                   * Wait for one of the previously defined conditions

* ### Bypass phase ###
bypass0:    ldcd rst ofs keep keep Tbypass c3; * Load in the counter 3 the length of the off_phase phase
            stos off off off;              * Turn VBAT off, BOOST off, LS off
            cwer hold0 tc3 row4;           * Jump to hold when tc3 reaches end of count
            wait row14;                    * Wait for one of the previously defined conditions
```

5.2 Channel 1 - Ucore1 - controls injectors 3 and 4

```
* ### Channel 1 - uCore1 controls the injectors 3 and 4 ###

* ### Variables declaration  ###

* Note: The data that defines the profiles are shared between the two microcores.

* ### Initialization phase ###
init1:      stgn gain12.6 sssc;              * Set the gain of the opamp of the current measure block 2
            ldjr1 eoinj1;                    * Load the eoinj line label Code RAM address into the register jr1
            ldjr2 idle1;                     * Load the idle line label Code RAM address into the register jr2
            cwef jr1 _start row1;            * If the start signal goes low, go to eoinj phase

* ### Idle phase- the uPC loops here until start signal is present ###
idle1:      joslr inj3_start start3;         * Perform an actuation on inj3 if start 3 (only) is active
            joslr inj4_start start4;         * Perform an actuation on inj4 if start 4 (only) is active
            jmpf jr1;                        * If more than 1 start active at the same time(or none), no actuation

* ### Shortcuts definition per the injector to be actuated ###
inj3_start: dfsct hs3 hs4 ls3;               * Set the 3 shortcuts: VBAT, VBOOST, LS
            jmpr boost1;                     * Jump to launch phase

inj4_start: dfsct hs3 hs4 ls4;               * Set the 3 shortcuts: VBAT, VBOOST, LS
            jmpr boost1;                     * Jump to launch phase

* ### Launch phase enable boost ###
boost1:     load Iboost dac_sssc _ofs;       * Load the boost phase current threshold in the current DAC
            cwer peak1 ocur row2;            * Jump to peak phase when current is over threshold
            stf low b0;                      * set flag0 low to force the DC-DC converter in idle mode
            stos off on on;                  * Turn VBAT off, BOOST on, LS on
            wait row12;                      * Wait for one of the previously defined conditions

* ### Peak phase continue on Vbat ###
peak1:      ldcd rst _ofs keep keep Tpeak_tot c1; * Load the length of the total peak phase in counter 1
            load Ipeak dac_sssc _ofs;        * Load the peak current threshold in the current DAC
            cwer bypass1 tc1 row2;           * Jump to bypass phase when tc1 reaches end of count
            cwer peak_on1 tc2 row3;          * Jump to peak_on when tc2 reaches end of count
            cwer peak_off1 ocur row4;        * Jump to peak_off when current is over threshold
            stf high b0;                     * set flag0 high to release the DC-DC converter idle mode

peak_on1:   stos on off on;                  * Turn VBAT on, BOOST off, LS on
            wait row124;                     * Wait for one of the previously defined conditions

peak_off1:  ldcd rst ofs keep keep Tpeak_off c2; * Load in the counter 2 the length of the peak_off phase
            stos off off on;                 * Turn VBAT off, BOOST off, LS on
            wait row123;                     * Wait for one of the previously defined conditions

* ### Bypass phase ###
bypass1:    ldcd rst ofs keep keep Tbypass c3; * Load in the counter 3 the length of the off_phase phase
            stos off off off;                * Turn VBAT off, BOOST off, LS off
            cwer hold1 tc3 row4;             * Jump to hold when tc3 reaches end of count
            wait row14;                      * Wait for one of the previously defined conditions

* ### Hold phase on Vbat ###
hold1:      ldcd rst _ofs keep keep Thold_tot c1; * Load the length of the total hold phase in counter 2
            load Ihold dac_sssc _ofs;        * Load the hold current threshold in the DAC
            cwer eoinj1 tc1 row2;            * Jump to eoinj phase when tc1 reaches end of count
            cwer hold_on1 tc2 row3;          * Jump to hold_on when tc2 reaches end of count
            cwer hold_off1 ocur row4;        * Jump to hold_off when current is over threshold

hold_on1:   stos on off on;                  * Turn VBAT on, BOOST off, LS on
            wait row124;                     * Wait for one of the previously defined conditions

hold_off1:  ldcd rst _ofs keep keep Thold_off c2; * Load the length of the hold_off phase in counter 1
            stos off off on;                 * Turn VBAT off, BOOST off, LS on
            wait row123;                     * Wait for one of the previously defined conditions

* ### End of injection phase ###
eoinj1:     stos off off off;                * Turn VBAT off, BOOST off, LS off
            stf high b0;                     * set flag0 to high to release the DC-DC converter idle mode
            jmpf jr2;                        * Jump back to idle phase

* ### End of Channel 1 - uCore1 code ###
```

5.3 Channel 2 - Ucore0 - DCDC control

```
* ### Channel 2 - uCore0 controls dc-dc ###

* ### Variables declaration  ###

#define Vboost_high 0;                  * The Vboost_high voltage value is stored in the Data RAM address 0
#define Vboost_low 1;                   * The Vboost_low voltage value is stored in the Data RAM address 1
#define Isense4_high 2;                 * The Isense4_high current value is stored in the Data RAM address 2
#define Isense4_low 3;                  * The Isense4_low current value is stored in the Data RAM address 3

* ### Initialization phase ###
init0:      stgn gain5.8 ossc;          * Set the gain of the opamp of the current measure block 4
            load Isense4_low dac_sssc _ofs;   * Load Isense4_high current threshold in DAC 4L
            load Isense4_high dac4h4n _ofs;   * Load Isense4_high current threshold in DAC 4H
            stdm null;                  * Set the boost voltage DAC access mode
            cwer dcdc_idle _f0 row1;    * Wait table entry for Vboost under Vboost_low threshold condition
            cwer dcdc_on _vb row2;      * Wait table entry for Vboost under Vboost_low threshold condition
            cwer dcdc_off vb row3;      * Wait table entry for Vboost over Vboost_high threshold condition

* ### Asynchronous phase ###
dcdc_on:load Vboost_high dac4h4n _ofs;  * Load the upper Vboost threshold in vboost_dac register
        stdcctl async;                  * Enable asynchronous mode
        wait row13;                     * Wait for one of the previously defined conditions

* ### Synchronous phase ###
dcdc_off:load Vboost_low dac4h4n _ofs;  * Load the upper Vboost threshold in vboost_dac register
        stdcctl sync;                   * Enable synchronous mode
        wait row12;                     * Wait for one of the previously defined conditions

* ### Idle phase ###
dcdc_idle: stdcctl sync;                * Enable synchronous mode
           jocr dcdc_idle _f0;          * jump to previous line while flag 0 is low
           jmpr dcdc_on;                * force the DC-DC converter on when flag 0 goes high

* ### End of Channel 2 - uCore0 code ###
*
```

5.4 Channel 2 – Ucore1 – Fuel Pump Drive Source Code

```
* ### Channel 2 - uCore0 controls dc-dc ###


* ### Channel 2 - uCore1 drives fuel pump ###

* ### Variables declaration  ###

* Note: The data are stored into the dataRAM of the channel 1.
#define Ipeak 5;                                * The peak current value is stored in the Data RAM address 5
#define Ihold 6;                                * The hold current value is stored in the Data RAM address 6
#define Thold_off 7;                            * The hold off time is stored in the Data RAM address 7
#define Thold_tot 8;                            * The hold phase duration is stored in the Data RAM address 8
* Note: The Tpeak_tot variable defines the current profile time out. The active STARTx pin is expected to toggle in is low state before this time out.

* ### Initialization phase ###
init1:      stgn gain19.4 ossc;                 * Set the gain of the opamp of the current measure block 1
            ldjr1 eoact1;                       * Load the eoinj line label Code RAM address into the register jr1
            ldjr2 idle1;                        * Load the idle line label Code RAM address into the register jr2
            cwef jr1 _start row1;               * If the start signal goes low, go to eoinj phase

* ### Idle phase- the uPC loops here until start signal is present ###
idle1:      joslr act5_start start5;             * Perform an actuation on act5 if start 5 (only) is active
            joslr act6_start start6;             * Perform an actuation on act6 if start 6 (only) is active
            jmpf jr1;                           * If more than 1 start active at the same time(or none), no actuation

* ### Shortcuts definition per the injector to be actuated ###
act5_start: dfsct hs5 ls5 undef;                * Set the 2 shortcuts: VBAT, LS
            jmpr peak1;                         * Jump to launch phase

act6_start: dfsct hs5 ls6 undef;                * Set the 2 shortcuts: VBAT, LS
            jmpr peak1                          * Jump to launch phase

* ### Launch peak phase on bat ###
peak1:      load Ipeak dac_ossc _ofs;           * Load the boost phase current threshold in the current DAC
            cwer hold1 cur3 row2;               * Jump to peak phase when current is over threshold
            stos on on keep;                    * Turn VBAT off, BOOST on, LS on
            wait row12;                         * Wait for one of the previously defined conditions

* ### Hold phase on Vbat ###
hold1:      ldcd rst _ofs keep keep Thold_tot c1; * Load the length of the total hold phase in counter 2
            load Ihold dac_ossc _ofs;           * Load the hold current threshold in the DAC
            cwer eoact1 tc1 row2;               * Jump to eoinj phase when tc1 reaches end of count
            cwer hold_on1 tc2 row3;             * Jump to hold_on when tc2 reaches end of count
            cwer hold_off1 cur3 row4;           * Jump to hold_off when current is over threshold

hold_on1:   stos on on keep;                    * Turn VBAT on, LS on
            wait row124;                        * Wait for one of the previously defined conditions

hold_off1:  ldcd rst _ofs off on Thold_off c2;  * Load the length of the hold_off phase in counter 1 and turn VBAT off, LS on
            wait row123;                        * Wait for one of the previously defined conditions

* ### End of injection phase ###
eoact1:     stos off off keep;                  * Turn VBAT off, LS off
            jmpf jr2;                           * Jump back to idle phase

* ### End of Channel 2 - uCore1 code ###
```

6 Revision history

Revision history

Rev	Date	Description
v.3	20190429	• Table 29: added stmfm instruction • Section 3.2.2: added description for stmfm instruction
v.2	20190415	• Section 2.3: replaced "8 startx pins" by "6 startx pins" • Section 3.2.2 (ldca): updated the description for Off and On operand labels • Section 3.2.2 (ldcd): updated the description for Off and On operand labels
v.1	20171215	• Initial version

7 Legal information

7.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Suitability for use in automotive applications — This NXP Semiconductors product has been qualified for use in automotive applications. Unless otherwise agreed in writing, the product is not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

7.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

NXP — is a trademark of NXP B.V.

Tables

Figures

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 29 April 2019
Document identifier: PT2001SWUG