# DECA EEP1I Specification Sheet

v2024.3

## 1 Requirements

- Separate interrupt stack pointer (ISP) full/descending
- ISP is initialised to 0x0000
- Hardware interrupt entry pushes PC and Flags onto ISP stack, clears FlagI
- Hardware RET restores PC and Flags popping from ISP
- Hardware error (interrupt) if RET is executed with ISP = 0x0000
- Interrupt target comes from vector generated by a Vectored Interrupt Controller with 8 prioritised interrupt levels 0 - 7 and hardware priority control.
- 4 bit field in FLAGS, PRIO(3:0) = FLAGS((8-5), provides priority 0 – 8
  - 0 → all interrupts enabled
  - 8 → no interrupts enabled
  - PRIO → n → interrupt i enabled for i > n
- When an interrupt of priority i is executed PRIO is updated to (i+1). When RET is executed PRIO is set to its previous value (in FLAGS)
- Interrupt is accepted by cpu only if FlagI is high
- On entry of interrupt, flags and PC are pushed on to ISP stack through hardware
- When interrupt is requested, current instruction in MEMADDR does not execute
- On calling RET instruction, flags and PC are popped from ISP stack through hardware
- A Memory mapped Interval Timer block (TIMER) must have programmer's spec similar to that in the lectures. The block should have memory control interface `TIMEREN,AD0,WEN` as shown in Table 1.

| `TIMEREN` | `AD1` | `AD0` | Register read or written as `WEN` |
|---|---|---|---|
| 0 | X | X | none |
| 1 | 0 | 0 | `INTSTATUS` |
| 1 | 0 | 1 | `INTCNT` |
| 1 | 1 | 0 | `INTTIME` |
| 1 | 1 | 1 | none |

Table 1: Interrupt Timer Register Addressing

# 2 Implementation

- When IRQ is recived and accepted, the current instruction is not executed by the CPU rather on exit of interrupt, PC is returned to this instruction value to execute then.
- IRQ is allowed on all instructions except for EXT

## 2.1 INTC

Controls timing and execution of relevant instructions to handle interrupt entry and exit.

- ISTATE is a state machine that sets the timing for execution on interrupt entry and exit.
- When ISTATE gives a STALL signal, PC in the controlpath is not incremented for the next cycle.
- Instead the necessary instruction as determined by ISTATE is carried over to the datapath as INTINS(15:0) and replaces INS(15:0) through a multiplexer.
- INTC also outputs RET2 and IRQRESET which is high for the second cycle of interrupt exit only and is used by other blocks as detailed below.

### 2.1.1 ISTATE

State Machine to control timing and push and pop of flags and PC from ISP stack on entry and exit of interrupt.

1. Inputs
   - INTJUMPEN: Active when an IRQ is accepted and the jump to ISR is enabled It is active for one cycle and can be made high only if Flag I is high and control path is enabled. We ensure that it is active only for one cycle as INTJUMPEN sets FlagI to 0 for the next cycle until RETINT or SETI. INTJUMPEN is not allowed when there was an Extend instruction in the previous cycle.

$$INTJUMPEN = FLAGIQ \cdot CPEN \cdot IRQ \cdot \overline{EXTFF1} \tag{1}$$

   - RETINT: Active when RET function is called
2. Outputs
   - STALL: When active, stops the instruction in MEMADDR from executing and replaces it with INTINS from INTC. Also halts PCNEXT from incrementing.
   - PUSH PC: Executes push PC command onto ISP stack
   - PUSH FLAGS: Executes push FLAGS command onto ISP stack
   - POP FLAGS: Executes Pop FLAGS command from ISP stack

$$IN0 = INTJUMPEN + RETINT \tag{2}$$
$$IN1 = RETINT \cdot \overline{INTJUMPEN} \tag{3}$$
$$Q0^{+} = IN0 \cdot \overline{Q0} \cdot \overline{Q1} \tag{4}$$
$$Q1^{+} = IN0 \cdot IN1 \cdot \overline{Q0} \cdot \overline{Q1} \tag{5}$$

## 2.2 VIC

Generates a vector pointing to address of ISR when an IRQ of relevant priority is recieved. Also controls PRIO which determines the priority level of the current interrupt being executed

- Priority levels range from 0-8 where an interrupt is enabled if i ¡ PRIO.
- PRIO is initialized to 0 and returns back to 0 when no interrrupt is being handled

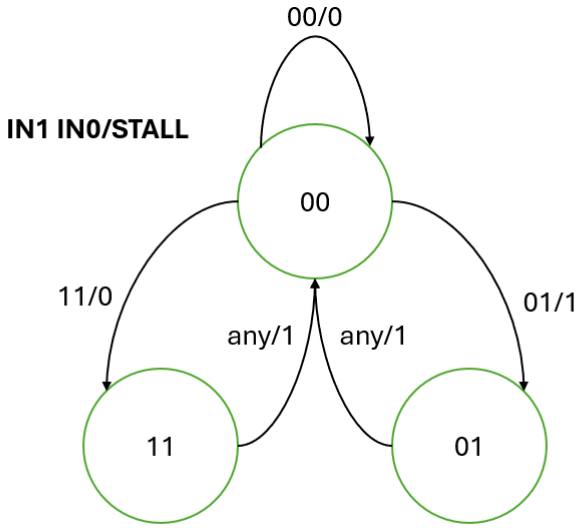


Figure 1: State Diagram (Mealy Model)

- PRIO (3) is INTEN that disables all interrupts if high
- There are 7 interupt levels ranging from 1 to 7 that is toggled through IRQn
- The IRQ is enabled if it is of the right priority as detaied above and if a IRQ of a higher priority is not high in the same cycle. Thus a max of one interrupt can be passed per cycle.
- The interrupt that is passed through will correspond to a relevant interrupt vector (INTVECTOR) that points to the relevant ISR through a multiplexer.
- if an interrupt is passed through, the IRQ toggle will go high.
- PRIONEXT(2:0) is PRIO(2:0) if IRQ is low and is the value of the interrupt that passed through if IRQ is high.
- PRIONEXT (3) is the value of the toggle INTEN
- PRIONEXT is stored in the RAM as a part of FLAGS on interrupt entry and is retireved on interrupt exit
- PRIO is PRIONEXT if IRQ is high
- PRIO from CPU is taken if RET is high (signifies end of previous ISR execution)

## 2.3 Stack Pointer (ISP)

**Push**

| EEP1 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| machine code | 0 | 0 | 0 | 0 | a | | | 0 | | SpOp | | 1 | 1 | 1 | (0) | |

**Pop**

| EEP1 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| machine code | 0 | 0 | 0 | 0 | 0 | | | 0 | | SpOp | | 1 | 0 | 1 | (0) | |

Figure 2: Stack Instructions

- SpOp decoding:

  - 0 = Register INS
  - 1 = PC INS
  - 2 = FLAGS INS
- The following are the decoded instructions achieved
  - POP REG : 0000 REG 00 00 101 00
  - PUSH REG : 0000 REG 00 00 111 00
  - POP FLAGS : 0000 000 00 10 101 00
  - PUSH FLAGS : 0000 000 00 10 111 00
  - POP PC : 0000 000 00 01 101 00
  - PUSH PC : 0000 000 00 01 111 00
- PUSH instructions increment ISP by one and POP instructions decrease ISP by one.
- PUSH uses post-increment value of ISP and POP uses pre-decrement value of ISP.
- The instructions are decoded through a separate block called INTDECODE
- INTDECODE outputs wether it was a PUSH or POP instruction and SPSELC which has the same purpose as SpOp.
- RETINT, SETI and CLRI remains the same.
- RETINT now calls POPPC and clears flag I to avoid being interrupted

## 2.4 Timer

Requirements and implementation as detailed in lecture slides.