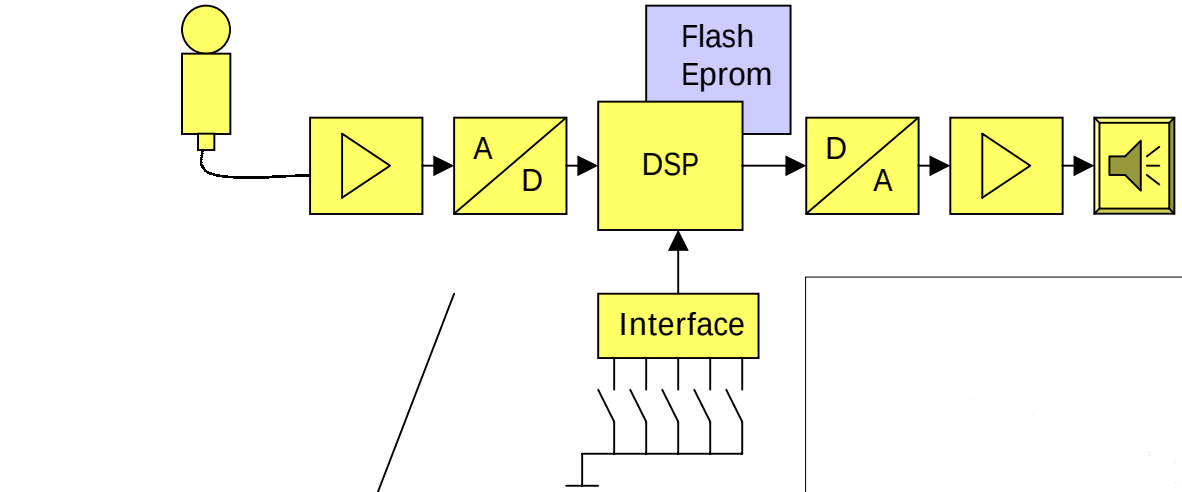
# Embedded DSP: Mini Project

## - Realization of a simple Digital Voice Recorder-

Labaratory Project: Real-Time Signal Processing with SHARC 21061

Objectives

- To become familiar with Visual DSP++ and the SHARC-EZ-KIT-Lite.
- Learn to program and use a control flow.
- Learn how to use interrupt driven I/O.
- Learn how to use I/O by polling mode.
- Learn to use Codec functions.




Voice Stick


SHARC-Kit Lite

A simple MCU controlled embedded system

# Embedded DSP: Mini Project

- The Voice Recorder should be able to store 3 records which means that 3 buffers are necessary.
- Two records use a DM-data buffer with max. 5600 data samples.
- One record use a data buffer in the PM-memory.
- The sampling rate should be 5.512 KHz (5512 samples/second).

## Data recording by FLAG1:

- Each record (1,2,3) automatically uses the corresponding DM-buffer or PM-buffer (1,2,3).
- The record in process automatically stops when the buffer end is reached.
- A record should be started with the pushbutton FLAG1, which means that FLAG1 is set as an input (refer to internal SHARC register ***mode2***).
  - *How to modify the register mode2 by C or Assembler ?*
  - *How to read the Input switch FLAG1 by polling mode ?*
  - *How to debounce the switch by software ?*
- The record should be indicated by LED FLAG3, which means that FLAG3 is set as an output (refer to internal SHARC register ***mode2***).
  - *How to modify the register mode2 by C or Assembler ?*
  - *How to set the FLAG3 output to a certain visble time (blink...) ?*

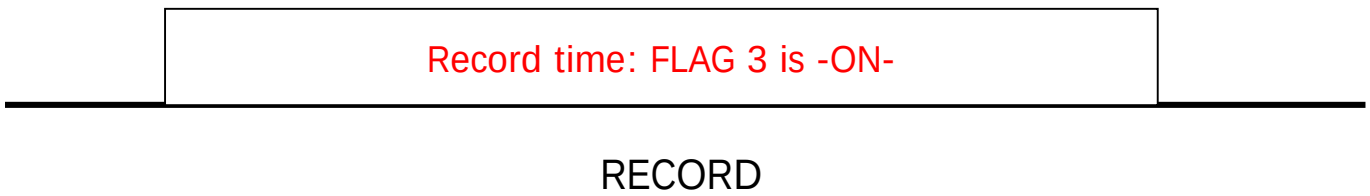
# Embedded DSP: Mini Project

Data recording by FLAG1:

Indications

- Start of record should be indicated by setting the LED/FLAG3 to -ON-.
- End of record should be indicated by clearing the LED/FLAG3 to -OFF-.
  - *How to set the FLAG3 output during record ?*

Optical indication of a record :



# Embedded DSP: Mini Project

Exercise: Development of a Digital Voice Recorder with the SHARC-Kit

**Play back by FLAG2**:

- Each record (1,2,3) should be started for play back by pushbutton FLAG2, which means that FLAG2 is set as an input.
  - *How to modify the register mode2 by C or Assembler ?*
  - *How to read the Input switch FLAG2 by polling mode ?*
  - *How to debounce the switch by software ?*
- The start and the end of a play back should be indicated by LED FLAG3, which means that FLAG3 is set to output (refer to internal SHARC register ***mode2***).
  - *How to modify the register mode2 by C or Assembler ?*
  - *How to set the FLAG3 output to a certain visble time (blink...) ?*
- Playing back of a selected record (1,2,3) should be indicated by a blink mode of LED/FLAG3.
  - *How to modify the register mode2 by C or Assembler ?*
  - *How to set the FLAG3 output to a certain visble time (blink...) ?*

# Embedded DSP: Mini Project

Exercise: Development of a simple Digital Voice Recorder with the SHARC-Kit

Optical indication of Play back:


Play back time: FLAG 3 is -ON-OFF-ON-OFF.....-

# Embedded DSP: Mini Project

## Selecting a buffer for recording or play-back mode:

- A consecutive record should be selected by the pushbutton IRQ1. At the start of the programm the first buffer is selected by default.
- The pushbutton IRQ1 is linked to the external interrupt IRQ1 of the SHARC.
- By pressing this pushbutton the next buffer should be automatically selected. By each pressing of the pushbutton the pointer for the next buffer should recirculate according to the count of the used number of records(1,2,3,1,2,3,...).
  - *How to use the IRQ1 by C or Assembler ?*
  - *How write and implement an interrupt service routine ?*

## Automatic record controlled by a programmed sound-level

- Controlled by a software switch
- If used the FLAG1 enables the record.
- The record will be started by crossing over the programmed threshold-level.

# Embedded DSP: Mini Project

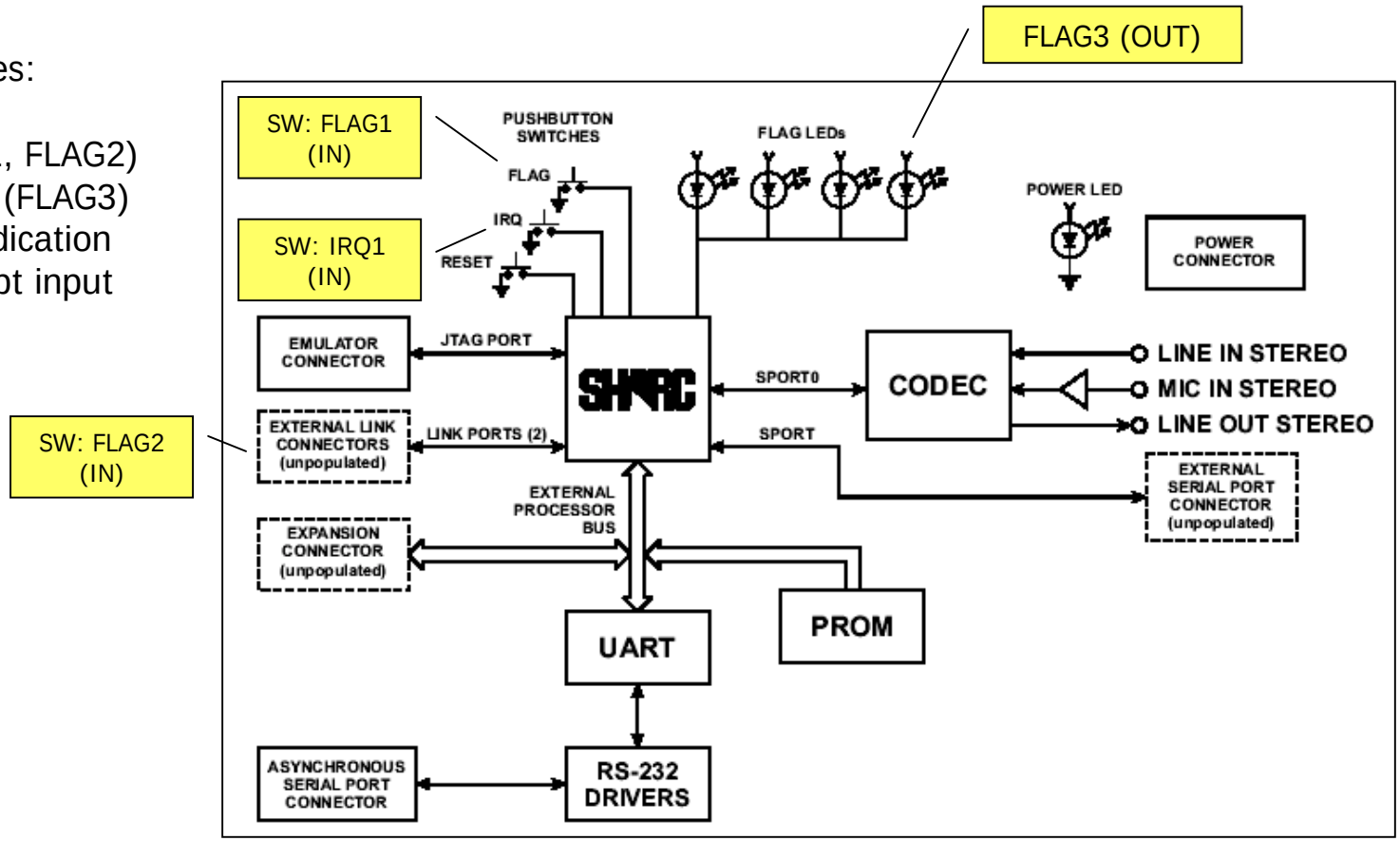
Control flow for the application:

- Start program
- Select buffer 1,2 or 3
- Select mode record or mode play-back
- Do it: Record or play back
- Wait for end of record or end of play-back
- Continue
- The program has to be embedded in the control flow of the CODEC-function.
- Codec: Read/Write by IRQ2, Reset by FLAG0
- The functionality of setting gain and sample rate as done in tt.c should be preseved.
- Take as a basic program the file tt.c from the demonstration software.
- The words from the CODEC are in 16 bit format. To increase the memory storage the data have to be packed in MSW and LSW format. Two words of the CODEC (16 Bit) are stored in one memory location (SHIFT-operation).

# Embedded DSP: Mini Project

Used ressources:

- FLAG inputs
  - (FLAG1, FLAG2)
- FLAG output (FLAG3)
  - LED indication
- IRQ1 interrupt input



# Embedded DSP: Mini Project

Steps of development:

- Create a new folder \recorder\ in the directory \demo\.
- Copy all files from \tt\ to \recorder\.
- Start Visual DSP++ and update the environment settings according to the new directory.
- Use the tt.c as the basic program for the new program.
- Modify in the file tt.ldf the target memory definition according to the requirements of the used buffer.
  - C-Compiler is realized by an Runtime Header Model which is connected to the Loader-Description-File for the target hardware system.
    - The following default segments from tt.ldf should be used; the DM memory area has to be modified.
      - seg_rth (PM memory: Interrupt table/runtime header)
      - seg_init (PM memory: code)
      - seg_pmco (PM memory: data)
      - seg_dmda (DM memory: data) (has to be modify for MiniProject !)
      - seg_heap (DM memory: heap space) (has to be modify for MiniProject !)
      - seg_stack (DM memory: stack space) (has to be modify for MiniProject !)
  - Decrease the dm memory size of stack and heap.
  - Increase the dm memory size for the 3 buffer !

# Embedded DSP: Mini Project

- Build a state diagram with the states record, play back and skip buffer.
- Develope the program and test the functionality on the EZ-KIT-Lite.

# Embedded DSP: SHARC programming

Control/Status Register of SHARC-processor

- Some of the registers are located in the processor core, called system registers.
- System register are a subset of the universal register.
- The core system register are:
  - MODE1, MODE2, ASTAT, STKY, IMASK, IMASKP, USTAT1 and USTAT2
- The remaining control registers are located in SHARCs I/O processor (IOP).
  - These includes the SYSCON and SYSTAT registers, which are memory-mapped in the internal memory.
- They can be written from an intermediate field in an instruction or they can be loaded from or stored to a data memory.
- They also can be transfered to or from any other universal register in one cycle.

# Embedded DSP: SHARC programming

System register (Core Processor)

- Are a subset of the universal register set.
- They can be written from an intermediate field in an instruction or they can be loaded from or stored to a data memory.
- They also can be transfered to or from any other universal register in one cycle.

| | | |
|---|---|---|
| (Project) | MODE1 | Mode control bits |
| (Project) | MODE2 | Mode control bits |
| | IRPTL | Interrupt latch |
| | IMASK | Interrrupt mask |
| | IMASKP | Interrupt mask pointer for nesting |
| (Project) | ASTAT | Arithmetic status register |
| | STKY | Sticky status flags |
| | USTAT1 | User-defined status flags |
| | USTAT2 | User-defines status flags |

- The *system register bit manipulation* instruction can be used to set, clear, toggle, or test specific bits in the system registers.
- An immediate field in the bit manipulation instruction specifies the affected bits.
- No transfer via the register file is necessary !

Bit manipulation <--> ALU/Shifter Bit manipulation

BIT SET register data <--> Rn=BSET Rx BY Ry |data
BIT CLR register data <--> Rn=BCLR Rx BY Ry |data
BIT TGL regsiter data <--> Rn=BTGL Rx BY Ry |data
BIT TST register data <--> BTST Rx BY Ry |data

- Examples:
  - BIT SET MODE2 0x000000070;
  - BIT TST ASTAT 0x000002000;

# Embedded DSP: SHARC programming

**MODE2:**

| Bit | Name | Definition | **Mini Project** (Modifications) |
|---|---|---|---|
| 0 | IRQ0E | 1=edge sensitive, 0=level sensitive | |
| 1 | IRQ1E | 1=edge sensitive, 0=level sensitive | **1** |
| 2 | IRQ2E | 1=edge sensitive, 0=level sensitive | |
| 3 | | Reserved | |
| 4 | CADIS | Cache disable | |
| 5 | TIMEN | Timer enable | **According to the program flow** |
| 6 | BUSLK | External bus lock (multiprocessor systems) | |
| 7-14 | | Reserved | |
| 15 | FLG0O | FLAG0 1=output, 0=input | |
| 16 | FLG1O | FLAG1 1=output, 0=input | **0** |
| 17 | FLG2O | FLAG2 1=output, 0=input | **0** |
| 18 | FLG3O | FLAG3 1=output, 0=input | **1** |
| 19 | CAFRZ | Cache freeze | |
| 20-27 | | Reserved | |
| 28-29 | | Silicon revision | |
| 30-31 | | Processor ID | |

# Embedded DSP: SHARC programming

**ASTAT:**

| Bit | Name | Definition | Mini Project (Modifications) |
|---|---|---|---|
| 0 | AZ | ALU result zero or floating-point underflow | |
| 1 | AV | ALU overflow | |
| 2 | AN | ALU result negative | |
| 3 | AC | ALU fixed point carry | |
| 4 | AS | ALU x input signs (ABS and MANT op.) | |
| 5 | AI | ALU floating-point invalid operation | |
| 6 | MN | Multiplier result negative | |
| 7 | MV | Multiplier overflow | |
| 8 | MU | Multiplier floating-point underflow | |
| 9 | MI | Multiplier floating-point invalid operation | |
| 10 | AF | ALU floating point operation | |
| 11 | SV | Shifter overflow | |
| 12 | SZ | Shifter result zero | |
| 13 | SS | Shifter input sign | |
| 14-17 | | Reserved | |
| 18 | BTF | Bit test flag for system register | |
| 19 | FLAG0 | FLAG0 Value | |
| 20 | FLAG1 | FLAG1 Value | **Input** |
| 21 | FLAG2 | FLAG2 Value | **Input** |
| 22 | FLAG3 | FLAG3 Value | **Output** |
| 23 | | Reserved | |
| 24-31 | CACC | | |

# Embedded DSP: SHARC programming

Programming FLAGs:(#include <21060.h>)

- File 21060.h contains basic:
  - definitions, functions and macros
  - for FLAG- and Timer functionality

- In C: Output function: set_flag(flagx, mode);

**Output examples in C:**

```
1) set_flag(SET_FLAG3, SET_FLAG);  /* Set Flag output state to 1 */
2) set_flag(SET_FLAG1, TGL_FLAG);  /* Toggles Flag1 (0>1 or 1>0)  */
3) set_flag8SET_FLAG1, CLR_FLAG); /* Clear Flag output state to 0 */
```

- In C: Input function: poll_flag_in(flagx, mode);

**Input examples in C:**

```
1) poll_flag_in(READ_FLAG1, FLAG_IN_LO_TO_HIGH);

2) poll_flag_in(READ_FLAG1, RETURN_FLAG_STATE);

3) while(1) {
      size=poll_flag_in(READ_FLAG2, RETURN_FLAG_STATE);
      if(size==0) break;
   }
```

**File: 21060.h**

```
.....
int set_flag(int _flag, int _mode);

#define SET_FLAG 0
#define CLR_FLAG 1
#define TGL_FLAG 2
#define TST_FLAG 3

#define SET_FLAG0 0
#define SET_FLAG1 1
#define SET_FLAG2 2
#define SET_FLAG3 3

int poll_flag_in(int _flag, int _mode);

#define READ_FLAG0 0
#define READ_FLAG1 1
#define READ_FLAG2 2
#define READ_FLAG3 3

#define FLAG_IN_LO_TO_HI 0
#define FLAG_IN_HI_TO_LO 1
#define FLAG_IN_HI 2
#define FLAG_IN_LO 3
#define FLAG_IN_TRANSITION 4
#define RETURN_FLAG_STATE 5
.....
```

# Embedded DSP: SHARC programming

Programming FLAGs as assembler inline code in C:(#include <def21060.h>)

- File def21060.h contains basic:
  - definitions, addresses and bit masks- and patterns
  - cover for all universal registers

**Flag output programming as assembler inline code for C environment:**

```
1)
asm(„#include <def21060.h>“);
asm(„bit set mode2 FLG3O;“); /* Set FLAG3 to output */
asm(„bit clr astat FLG3;“);              /* FLAG3 =0 */
2)
asm(„bit set mode2 FLG0O|FLG2O|FLG3O|IRQ1E;“);
```

**Flag testing input as assembler inline code for C environment:**

```
1)
asm(„#include <def21060.h>“);
asm(„bit tst astat 0x0008000;“); /* TF = 1 if FLG3 is ‚1‘  */
if TF .....;
2)
if FLAG2_IN ..compute..;  /* FLAG2_IN: condirtion code: (if,do) */
if NOT FLAG2_IN ..compute..;  /* FLAG2_IN: condirtion code: (if,do) */
```

File: def21060.h

```
...
...
/* MODE2 register */
#define IRQ0E    0x00000001      /* Bit  0: IRQ0- 1=edge sens. 0=level sens. */
#define IRQ1E    0x00000002      /* Bit  1: IRQ1- 1=edge sens. 0=level sens. */
#define IRQ2E    0x00000004      /* Bit  2: IRQ2- 1=edge sens. 0=level sens. */
#define CADIS    0x00000010      /* Bit  4: Cache disable */
#define TIMEN    0x00000020      /* Bit  5: Timer enable */
#define BUSLK    0x00000040      /* Bit  6: External bus lock */
#define FLG0O    0x00008000      /* Bit 15: FLAG0 1=output 0=input */
#define FLG1O    0x00010000      /* Bit 16: FLAG1 1=output 0=input */
#define FLG2O    0x00020000      /* Bit 17: FLAG2 1=output 0=input */
#define FLG3O    0x00040000      /* Bit 18: FLAG3 1=output 0=input */
#define CAFRZ    0x00080000      /* Bit 19: Cache freeze */
...
...
/* ASTAT */
...
#  define FLG0    0x00080000
#  define FLG1    0x00100000
#  define FLG2    0x00200000
#  define FLG3    0x00400000
...
...
```

# Embedded DSP: SHARC programming -Interrupt-

What are interrupts ?

- How programming interrupts ?
- Using a CODEC with interrupts

- Interrupts is an event that causes processor to halt what it is actually doing, and execute an interrupt service routine (ISR).
- An interrupt forces a subroutine call to a predefined address, the interrupt vector.
- SHARC-DSP assigns a unique vector to each type of interrupt.
- Interrupts are caused by a variety of conditions, both internal and external to the processor:
  - Timers
  - External interrupts
  - Internal interrupts: DMA (direct memory access)
- A interrupt is not recognized if it is not masked to state -ON-.
- In nested mode a priorization list schedules which interrupt is actually serviced.

# Embedded DSP: SHARC programming -Interrupt-

SHARC core processor cannot service an interrupt unless it is executing instructions or is in the special IDLE mode. IDLE is an instruction that halts the processor core until an external interrupt or the timer interrupt occurs. Interrupt service routines end with a RTI.

- To process an interrupt, the SHARC program sequencer performs the following actions:
  - Outputs the linked vector address.
  - Pushes the current PC value (return address) on the stack.
  - If the the interrupt source is one of the external IRQs or the VIRPT-IRQ (Multiprocessing), the sequencer pushes the current value of the ASTAT and MODE1 registers onto the status stack.
  - Set the appropriate bit in the interrupt latch register (IRPTL).
  - Achanges the interrupt mask pointer (IMASKP) to reflect the current interrupt nesting state. The nesting mode (NESTM) bit in the MODE1 register determines whether all interupts or only lower priority interrupts are masked during the servicing routine.
- At the end of an interrupt service routine, the RTI instruction causes the following actions.
  - Returns to the address, which was stored at the top of the stack.
  - Pops this value off of the stack.
  - Pops the status stack if the ASTAT and MODE1 status registers were pushed (see above).
  - Clears the appropriate bit in the interrupt latch register (IRPTL) and interrupt mask pointer (IMASKP).

# Embedded DSP: SHARC programming -Interrupt-

**Register: IRPTI&IMASK bit-position**

| Bit | Vector Address* | Interrupt Name | Function | |
|---|---|---|---|---|
| 0 | 0x00 | – | *reserved* | |
| 1 | 0x04 | RSTI | Reset (read-only)** | HIGHEST PRIORITY |
| 2 | 0x08 | – | *reserved* | |
| 3 | 0x0C | SOVFI | Status stack or loop stack overflow or PC stack full | |
| 4 | 0x10 | TMZHI | Timer=0 (high priority option) | |
| 5 | 0x14 | VIRPTI | Vector Interrupt | |
| 6 | 0x18 | IRQ2I | $\overline{\text{IRQ2}}$ asserted | |
| 7 | 0x1C | IRQ1I | $\overline{\text{IRQ1}}$ asserted | |
| 8 | 0x20 | IRQ0I | $\overline{\text{IRQ0}}$ asserted | |
| 9 | 0x24 | – | *reserved* | |
| 10 | 0x28 | SPR0I | DMA Channel 0 – SPORT0 Receive | |
| 11 | 0x2C | SPR1I | DMA Channel 1 – SPORT1 Receive (or Link Buffer 0) | |
| 12 | 0x30 | SPT0I | DMA Channel 2 – SPORT0 Transmit | |
| 13 | 0x34 | SPT1I | DMA Channel 3 – SPORT1 Transmit (or Link Buffer 1) | |
| 14 | 0x38 | LP2I | DMA Channel 4 – Link Buffer 2 | |
| 15 | 0x3C | LP3I | DMA Channel 5 – Link Buffer 3 | |
| 16 | 0x40 | EP0I | DMA Channel 6 – Ext. Port Buffer 0 (or Link Buffer 4) | |
| 17 | 0x44 | EP1I | DMA Channel 7 – Ext. Port Buffer 1 (or Link Buffer 5) | |
| 18 | 0x48 | EP2I | DMA Channel 8 – Ext. Port Buffer 2 | |
| 19 | 0x4C | EP3I | DMA Channel 9 – Ext. Port Buffer 3 | |
| 20 | 0x50 | LSRQI | Link Port Service Request | |
| 21 | 0x54 | CB7I | Circular Buffer 7 overflow | |
| 22 | 0x58 | CB15I | Circular Buffer 15 overflow | |
| 23 | 0x5C | TMZLI | Timer=0 (low priority option) | |
| 24 | 0x60 | FIXI | Fixed-point overflow | |
| 25 | 0x64 | FLTOI | Floating-point overflow exception | |
| 26 | 0x68 | FLTUI | Floating-point underflow exception | |
| 27 | 0x6C | FLTII | Floating-point invalid exception | |
| 28 | 0x70 | SFT0I | User software interrupt 0 | |
| 29 | 0x74 | SFT1I | User software interrupt 1 | |
| 30 | 0x78 | SFT2I | User software interrupt 2 | |
| 31 | 0x7C | SFT3I | User software interrupt 3 | LOWEST PRIORITY |

* Offset from base address: 0x0002 0000 for interrupt vector table in internal memory, 0x0040 0000 for interrupt vector table in external memory

** Non-maskable

Mini Project !

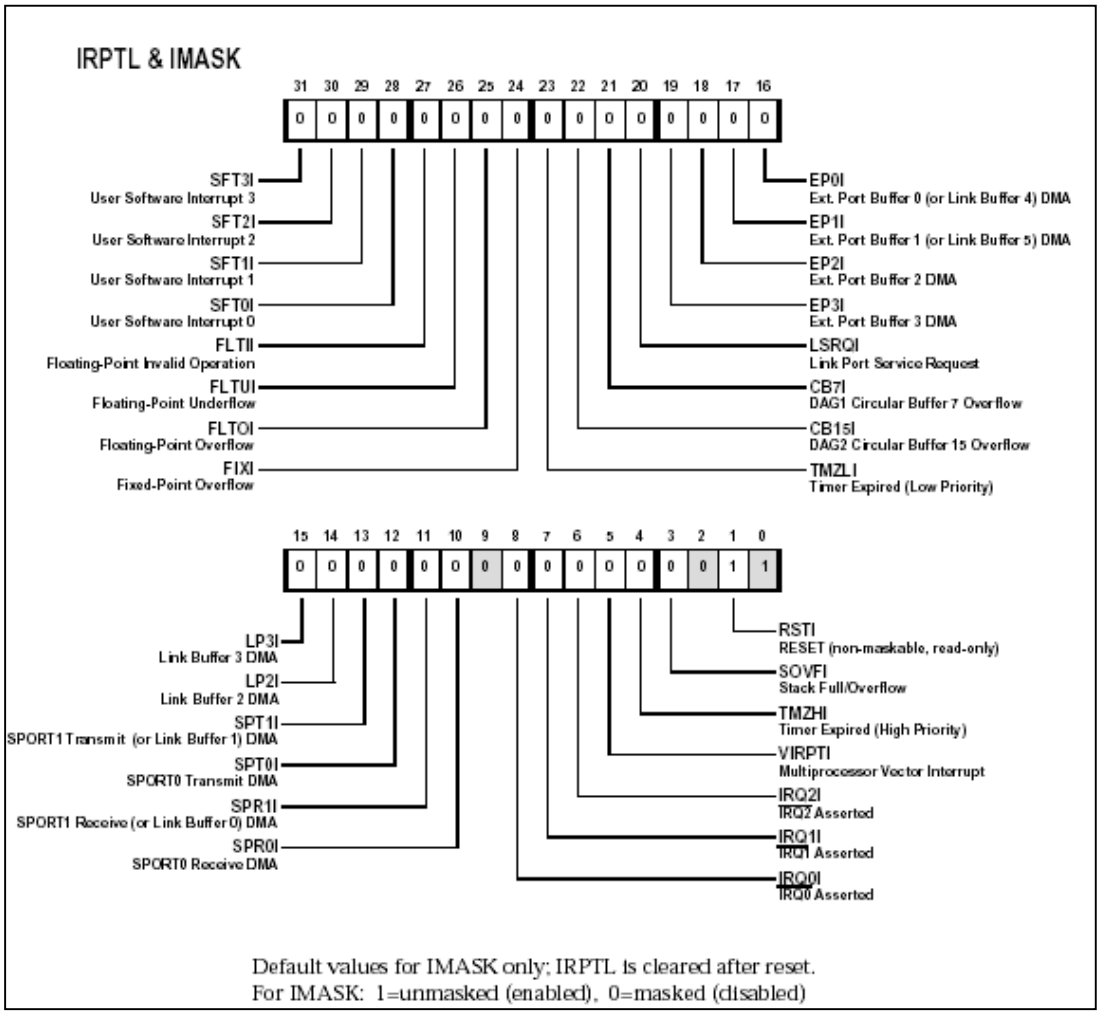
**Interrupt Vector Table:**

- Each is separated by 4 memory locations.
- Represents an offset from a base address.
- Internal memory:
  - base: 0x00020000
  - at beginning of block 0
- External memory:
  - base: 0x00400000

**(def21060.h)**

# Embedded DSP: SHARC programming

**Register: IRPTl&IMASK**



# Embedded DSP: SHARC programming -Interrupt-

Interrupt example in C environment via functions:

- Enable Sport 0 transmit and receive interrupts
- See also the examples from the demo software SHARC-EZKIT-Lite.
- The used functions are defined in the file signal.h Visual DSP++.
- Nested mode is defined in register mode1.
- Example for Enabling interupt nesting:

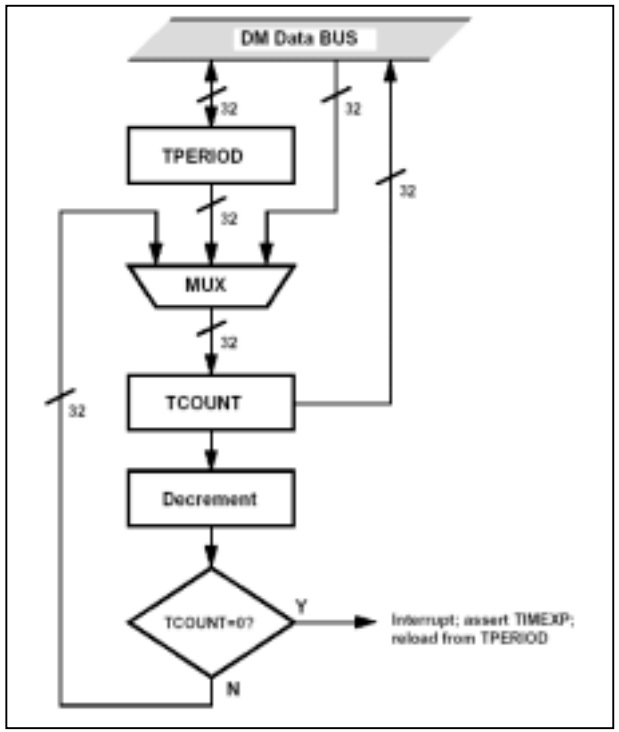
```
.....
asm(„include <def21060.h>“);
asm(„bit set mode1 NESTM“;);
.....
```

```
/* Sport 0 Receive Interrupt Service Routine */
void Sport0RcvIsr()
{
sport0_receive_flag = 1;
}
/* Sport 0 Transmit Interrupt Service Routine */
void Sport0TrsIsr()
{
sport0_transmit_flag = 1;
}
...
main()
{
..
   setup_sport0(); /* set sport0*/
   interrupt (SIG_SPR0I, Sport0RcvIsr);
   interrupt (SIG_SPT0I, Sport0TrsIsr);
   start_sport0_trs();
   start_sport0_rcv();
   ...
   while(1) {
      if(sport0_transmit_flag==1) {
         sport0_transmit_flag=0;
         ....
      }
      if(sport0_receive_flag==1) {
         sport0_transmit_flag=0;
         ....
      }
   }
}
```

# Embedded DSP: SHARC programming -Timer-

- The SHARC includes a programmable interval timer.
- The timer can generate periodic interrupts.
- The timer is controlled via the system register MODE2.
- The two registers TCOUNT and TPERIOD control the timer interval.
- The register TCOUNT contains the timer counter.
- The timer decrements the TCOUNT register by each clock cycle.
- When TCOUNT reaches zero, the timer generates an interrupt and asserts the TIMEXP pin on the chip.
- The TPERIOD register value specifies the frequency of timer interrupts.
- The number of cycles between interrupts is (TPERIOD + 1).
- The maximum value is $2^{32}$-1, so if the clock is 25 ns, the maximum time interval is 107,375 seconds.

| Register | Function | Width |
|---|---|---|
| TPERIOD | Timer Period Register | 32 bits |
| TCOUNT | Timer Counter Register | 32 bits |



# Embedded DSP: SHARC programming -Timer-

Timer Enable/Disable:

- To start and stop the timer, the bit TIMEN bit in the register mode2 has to beset or cleared.
- With the timer disabled, a new load of the TCOUNT register can be perfomred as an initial value.
- The value for the time interval has to be loaded in the register TPERIOD.
- Start the timer by setting the bit TIMEN.
- At reset the timer is always disabled.

Timer interrupts:

- When the value of TCOUNT reaches zero, the timer generates two interrupts.
- One with a relatively high priority
- One with a low priority.
- At reset both are masked out.
- Interrupt priority determines which interrupt is serviced first, when two ocuur in the same cycle.
- When nesting is enabled, only higher priority interrupts can interrupt a service routine in progress.

# Embedded DSP: SHARC programming -Timer-

See also the examples tt.c and blink.c from the demo software SHARC-EZKIT-Lite.

| PROGRAM CONTROL | |
|---|---|
| abort | abnormal program end |
| calloc | allocate / initialize memory |
| free | deallocate memory |
| idle | processor idle instruction |
| interrupt | define interrupt handling |
| poll_flag_in | test input flag |
| set_flag | sets the processor flags |
| timer_off | disable processor timer |
| timer_on | enable processor timer |
| timer_set | initialize processor timer |

Example in C:

```
/* Periodic time interrupt service routine */
void timer_high_priority(int sig_num)
{
        sig_num=sig_num;
        set_flag(SET_FLAG3, TGL_FLAG)
        timer_flag = 1;
}

main
{
        ....
        timer_off();                                    /* disable timer */
        timer_set(1000000,200000);                      /* program timer */
        Interrupt(SIG_TMH, timer_high_priority);        /* enable timer IRQ */
        ....
        timer_on();                                     /* start timer */
        ....
        while(1) {
                if(timer_flag=1) {
                    timer_flag=0;
                    .....
        }
}
```

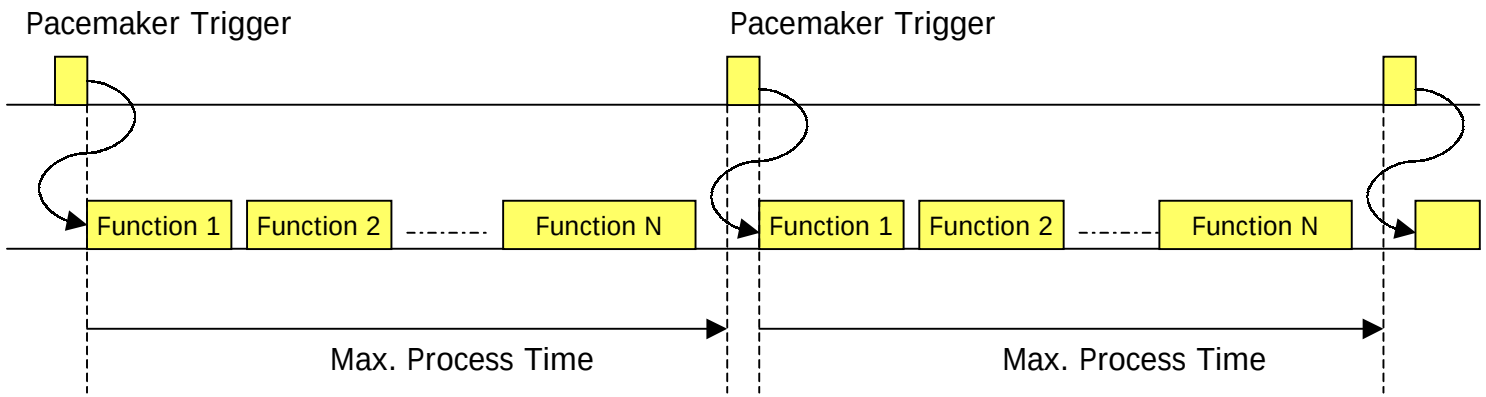
# TT.C

Discussion of the demo program tt.c
by overhead projector

# Embedded DSP: Paced Loop

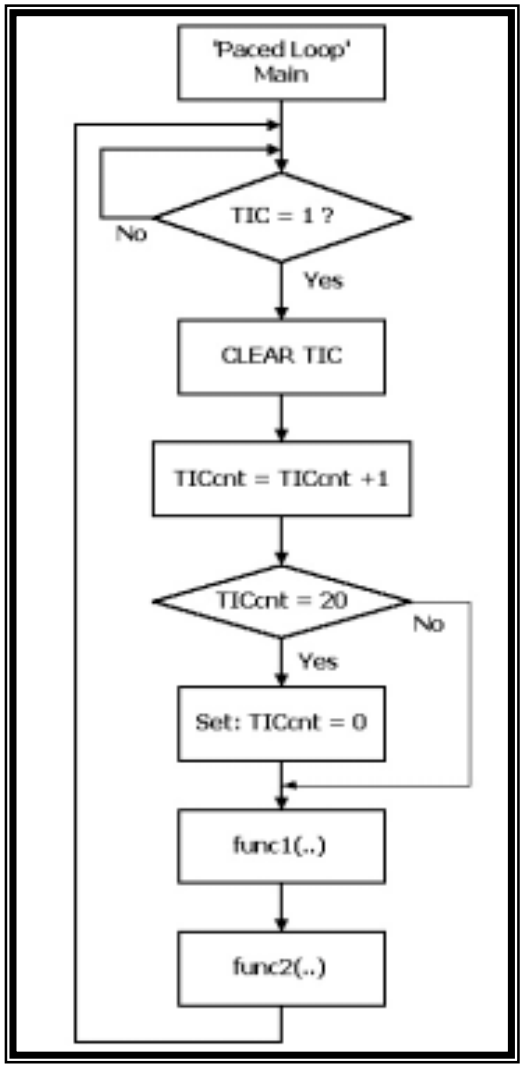
- The paced loop is a general purpose software structure that is suitable for a wide range of MCU/DSP applications.
- The main idea is to break the complete application into a series of tasks, such as:
  - reading data
  - processing data
  - storing results
  - reading system inputs
  - updating system outputs
- Each task is written as a function (subroutine).
- A main loop is realized out of „jump to subroutine“ instruction for each of the tasks.
- At the top of the loop is a software pacemaker integrated.
- When the pacemaker triggers, the list of task functions is executed one time and a branch instruction leads to the top of the loop to wait for the next pacemaker trigger.

# Embedded DSP: Paced Loop



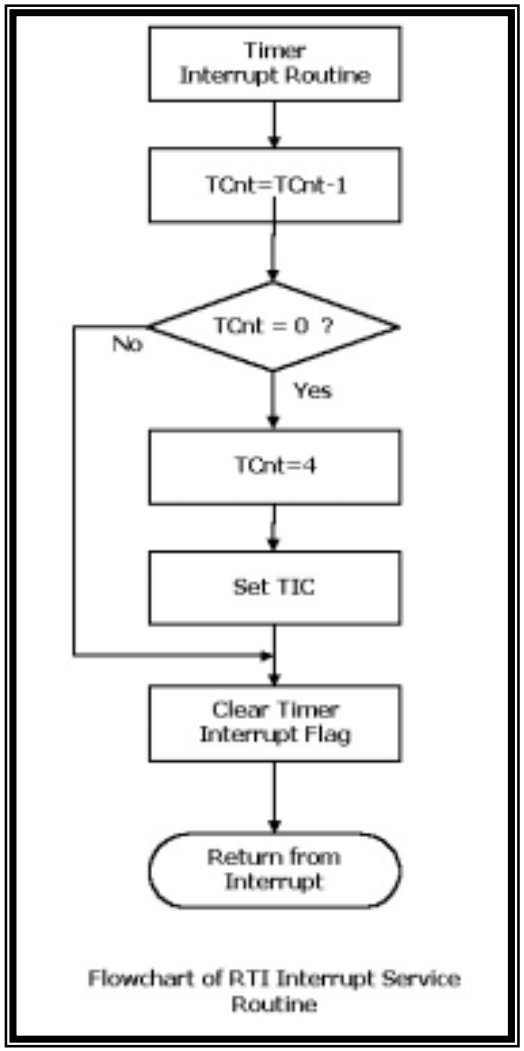

Pacemaker Trigger: Timer controlled by software

# Embedded DSP: Paced Loop




- The top block is a loop that waits for the pacemaker trigger TIC.
- The next blocks manages a counter TICcnt.
- When the counter TICcnt reachs a limit TICcnt will be cleared.
- This example acts with two functions (func1 and func2).
- The limitation of the number of tasks is given by the condition, that all tasks must finish quickly enough in order that no trigers are lost.
- The last block in the flowchart is just a branch to the top of the loop to wait for the next pacemaker trigger.

# Embedded DSP: Paced Loop



Flowchart of RTI Interrupt Service Routine

- The pacemaker loop is realized on a real time interrupt, called TimerIRQ.
- TimerIRQ is programmed to generate an interrupt to the CPU every K processor cycles.
- The flowchart shows what acts at each TimerIRQ interrupt.
- The interrupt is working asynchronously in respect to the main program.
- The variable TIC is used as a flag to tell the main program when it is time to increment the variable TICcnt and to step one time through the paced loop.
- In this example a variable Tcnt is used to count 4 real time interrupts before setting TIC to one.
- The main program watchs TIC to see when TIC becomes set.
- Every K processor cycles the timer interrupt flag will get set, trigering a timer interrupt request.
- One task of the interrupt service routine is to clear the flag that is responsible for the interrupt before returning from the interrupt.
- If the timer interrupt flag is not cleared before return, a new interrupt is generated immediately instead of waiting the next K time steps.

# Embedded DSP: Paced Loop

- The variable TICcnt is important for the pacemaker.
- TICcnt counts in this example from 0 to 20.
- As TICcnt increments from 19 to 20 , the program watches this and resets TICcnt within the pacemaker itself.
- TICcnt appears to count ftom 0 to 19; FLAG is equal to 0 on every twenth trigger of the pacemaker.
- Function#1 is the first task in the main loop and maintains a slower clock called TOC, which is incremented each time the paced loop executes and TICcnt is 0 (every 20. Pass through the loop).
- TOC is a software counter that counts from 0 to K.
- The task function#2 can use the current values of TICcnt and TIC to decide which action needs to be done on this pass activated by the paced loop.

# Embedded DSP: Paced Loop

- Some restrictions on the task functions:
- Each task function should do everything as quickly as possible.
- The total time required to execute one pass through all of the task functions must be less than 2 pacemaker triggers.

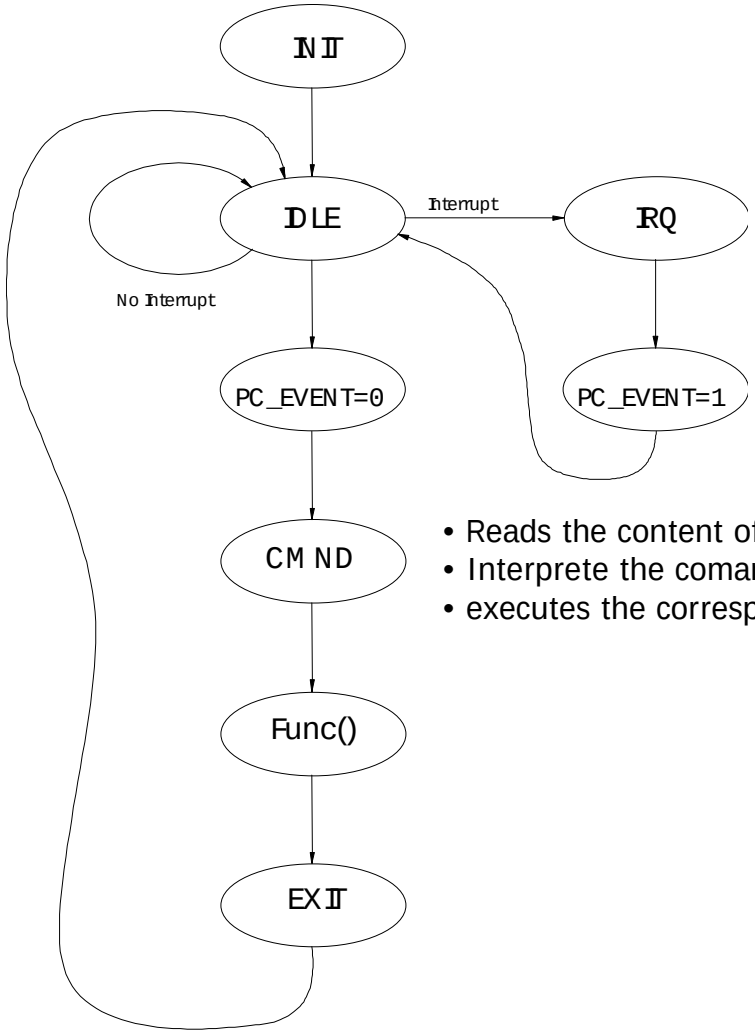
# Embedded DSP: Dispatcher

More complicated example:

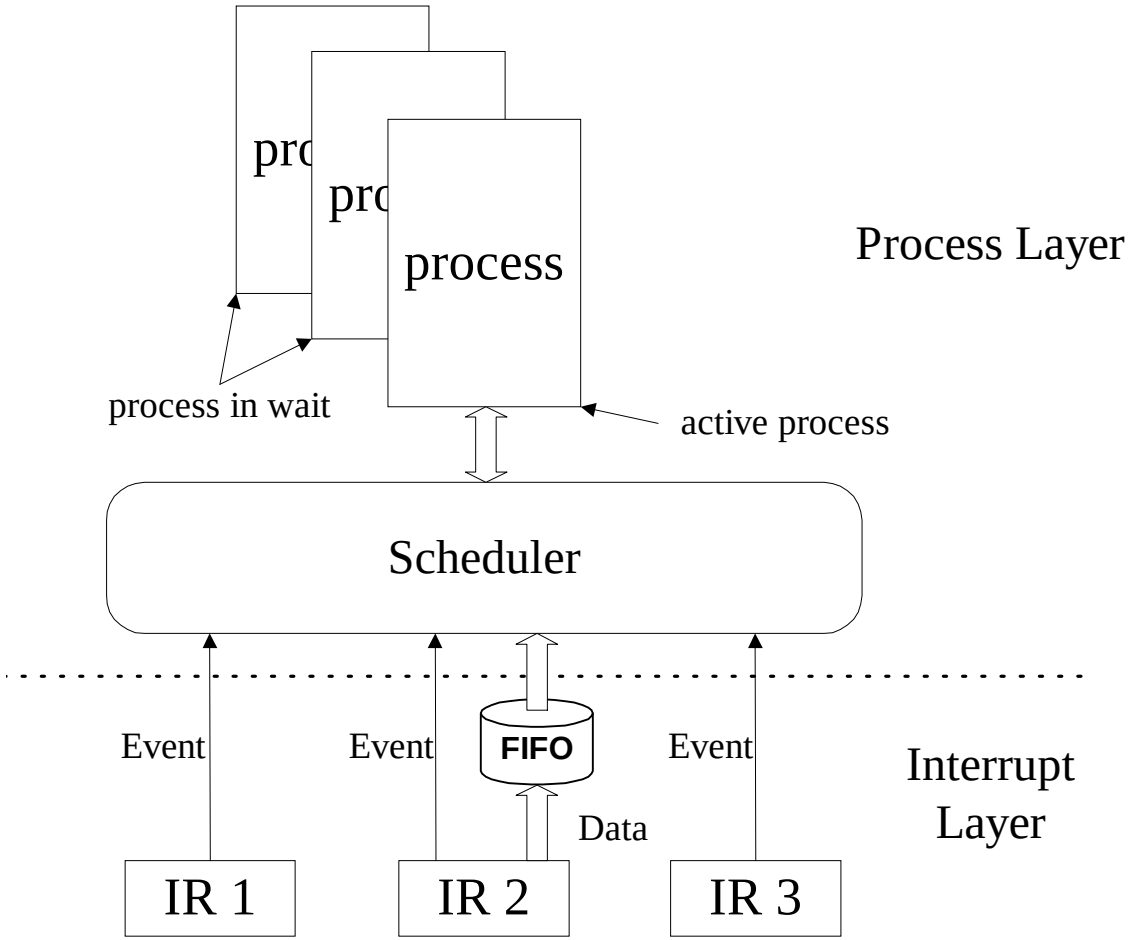
- Interrupt driven command dispatcher
- Interrupt routine: irq0_handler sets the event to 1 and reads a command from a communication channel.
- The switch loop peforms the corresponding comand driven function.

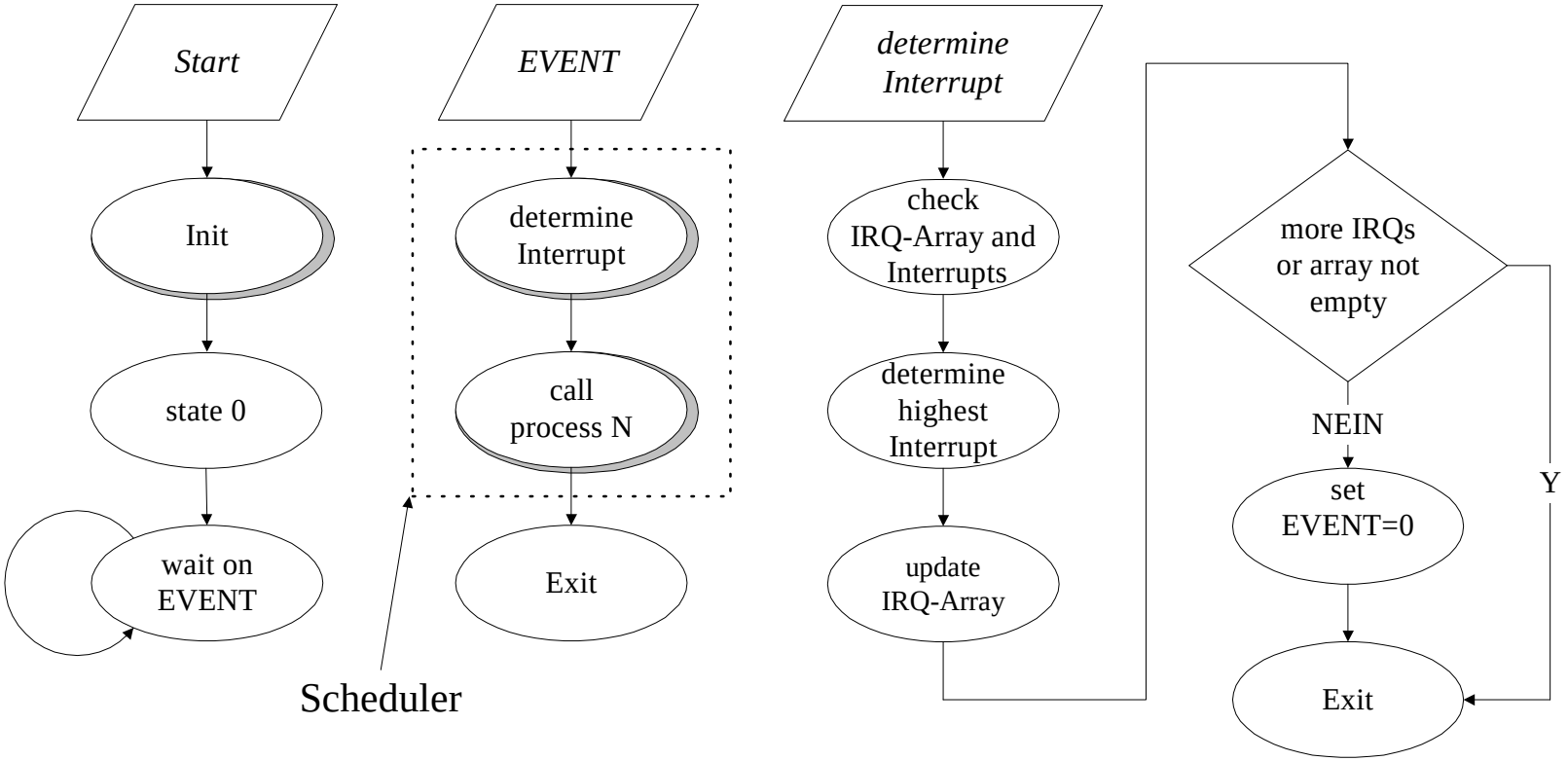
```
interrupt(SIG_IRQ0,irq0_handler);
..
..
..

 while(1)
 {
  idle();
  if (EventPC==1)
            {
             EventPC=0;
             switch(buffer.msg.cmd)
              {
               case LOAD_KOEFF:
                        ...
               case WAVELET:
                        ...
               case DATAPC_DSP:
                        ...
               case FFT:
                        ...
               case EXTERN:
                        ...
               case EXIT:
                        ...
            }
}
```

# Embedded DSP: Dispatcher




- Reads the content of a buffer
- Interprete the comand.
- executes the corresponding function.


process
process in wait
active process
Process Layer
Scheduler
Event
Event
FIFO
Event
Data
Interrupt Layer
IR 1
IR 2
IR 3


Start
Init
state 0
wait on
EVENT
EVENT
determine
Interrupt
call
process N
Exit
Scheduler
determine
Interrupt
check
IRQ-Array and
Interrupts
determine
highest
Interrupt
update
IRQ-Array
more IRQs
or array not
empty
NEIN
Y
set
EVENT=0
Exit