Enabling better device interaction with accelerometer

3 Feb 2013

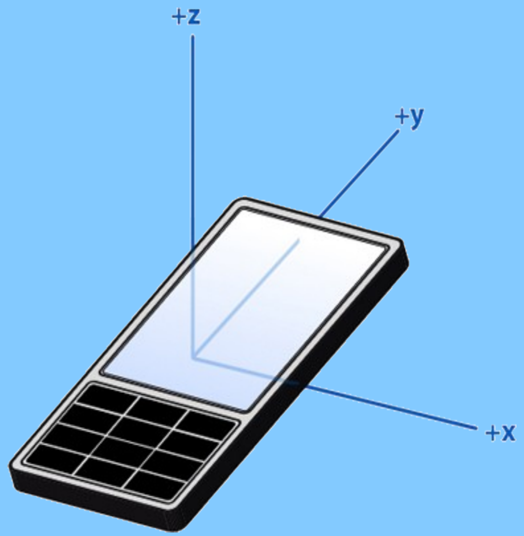
FOSDEM '13

etezian.org

Andi Shyti
Mika Laaksonen

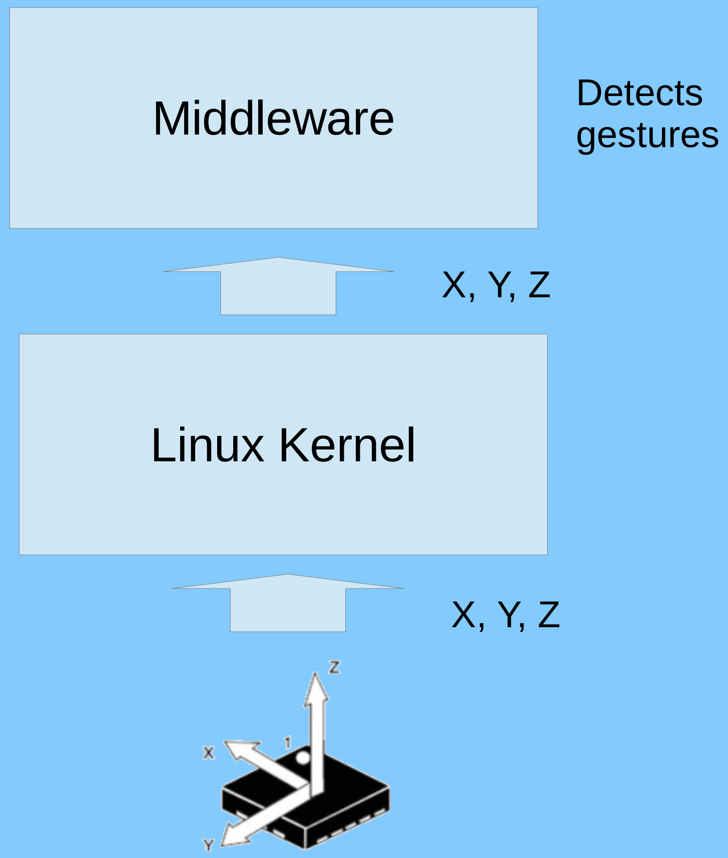
3D Accelerometer

- An accelerometer is a device which recognizes the gravitational field and acceleration on three axis X, Y and Z
- Use of it
  - Detect movements (or acceleration)
  - Orientation
  - Detect gestures



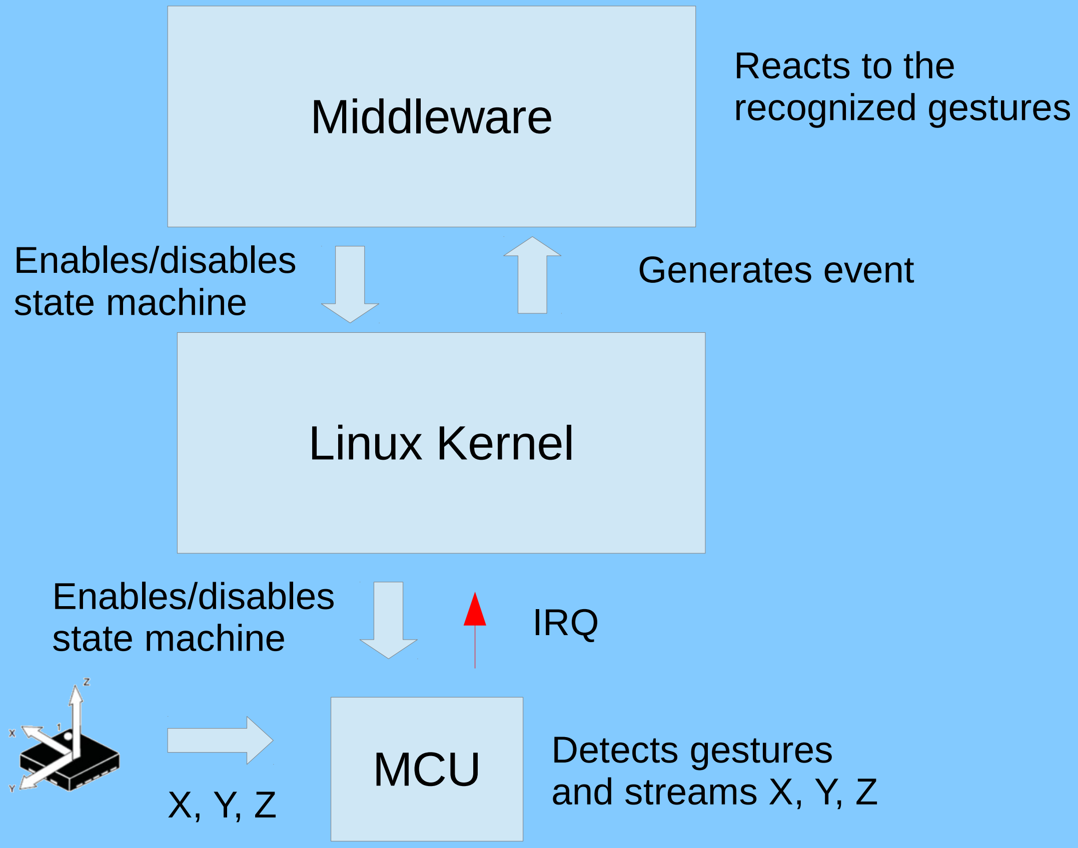
Gesture recognition (solution 1)

- Middleware software performs state machines for gesture recognition by reading the XYZ data flow with a specific frequency (in Hz)



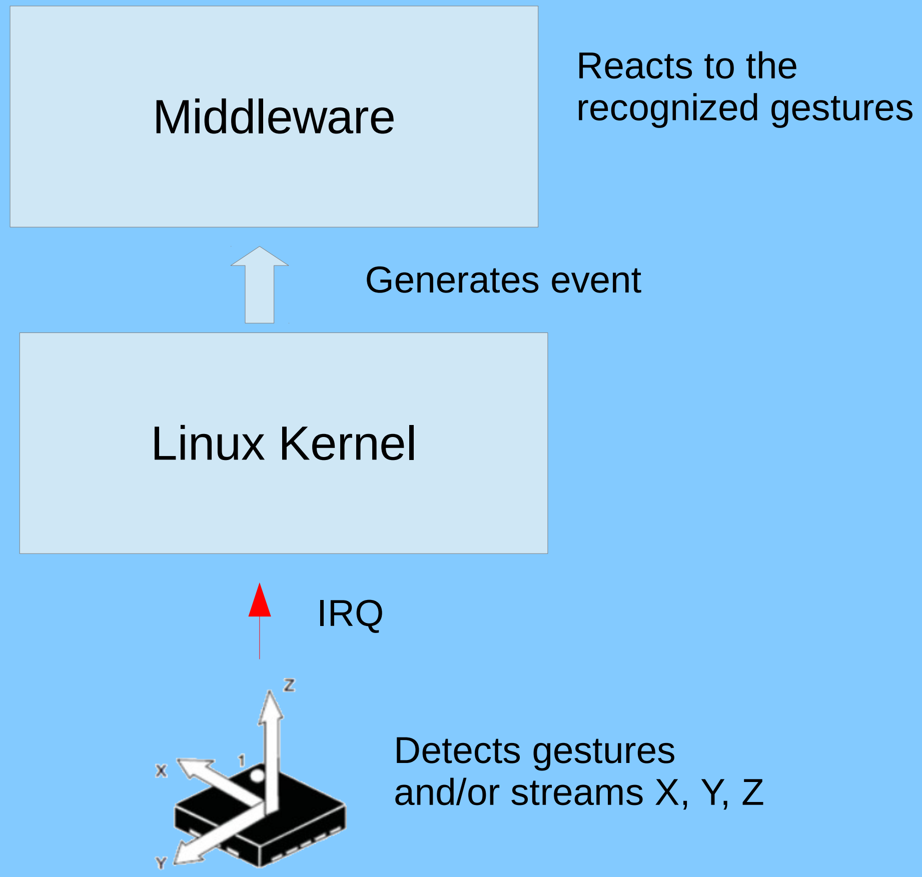
Gesture recognition (solution 2)

- An MCU device is placed as a “man in the middle”



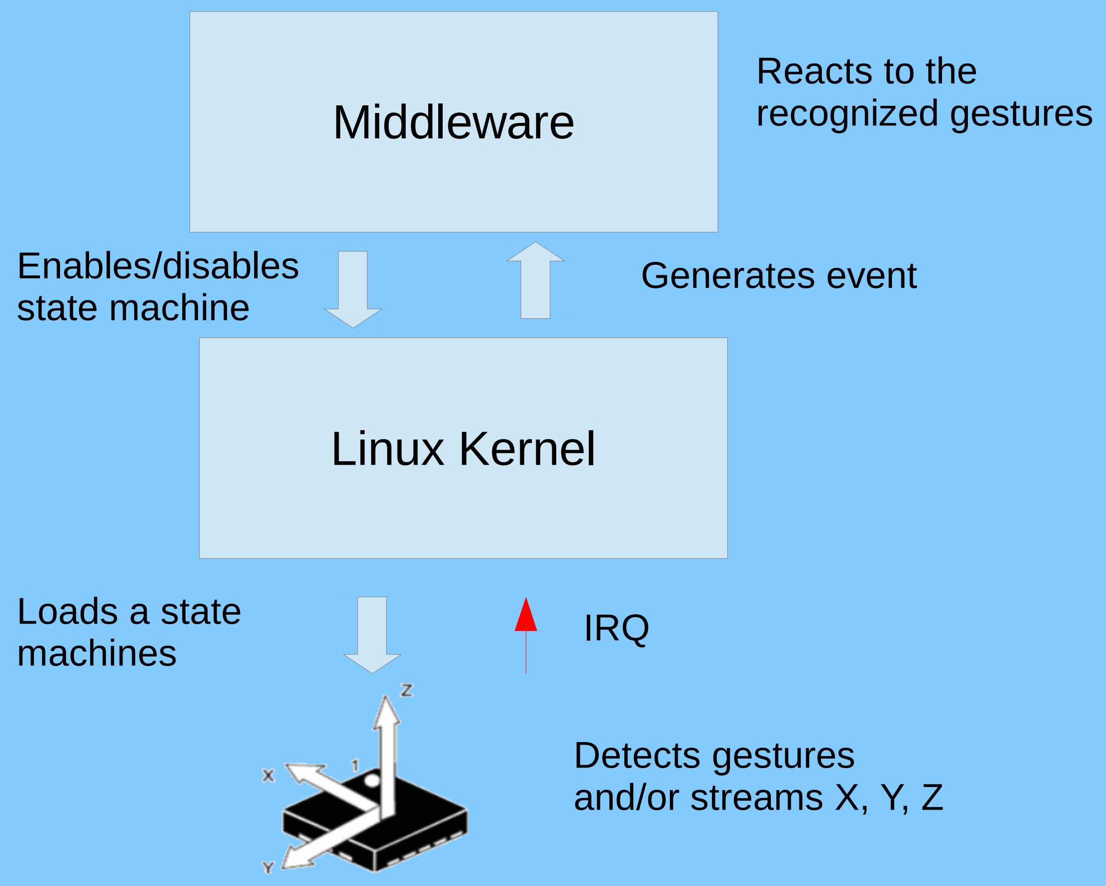
Gesture recognition (solution 3)

- A specific gesture can be recognized on the Accelerometer itself



Gesture recognition (solution 4)

- Middleware software chooses the gesture to be detected



Gesture recognition

- We like solution 4 because:
  - Less power consumption
  - User space software has good flexibility
  - Low memory requirement
  - Easy to implement

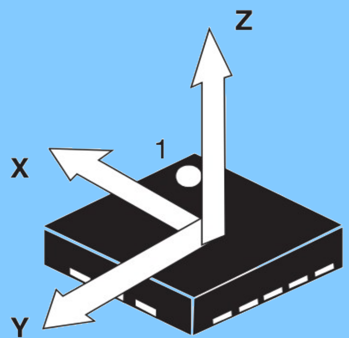
- One drawback:
  - The driver is a mess!

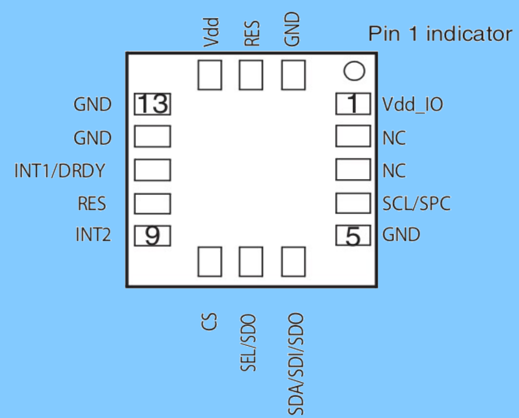
What's on the market

- Two devices
  - Kionix: KXCNL
    - http://www.kionix.com/accelerometers/kxcnl
  - STMicroelectronics: LIS3DSH
    - http://www.st.com/internet/analog/product/252716.jsp

The device

- The lis3dsh/kxcnl is a 3D accelerometer





- Two banks of registers for programmable patterns are available
  - Finite state machines is selected for pattern recognition

The device

- Wide range of frequencies available

3.125Hz	320ms
6.25Hz	160ms
12.5Hz	80ms
25Hz	40ms
50Hz	20ms
100Hz	10ms
400Hz	2ms
1600Hz	~0ms (ultra speed)

- Ultra low power consumption from 50µA at 3.125Hz to 250µA at 1600Hz

Interrupt logic

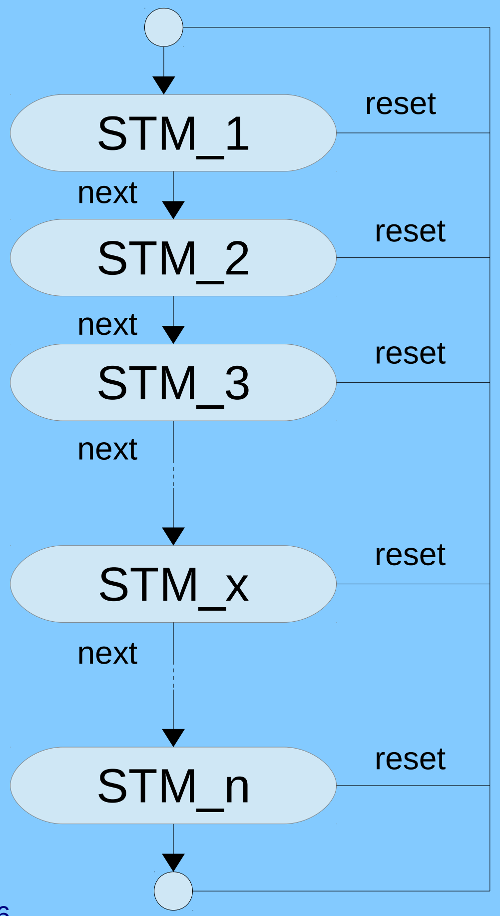
- Two interrupt lines
  - Active low / active high
  - Pulsed / latched
  - Inactive (high impedence)
- Interrupt are used to
  - Data ready (only INT1)
  - Signal pattern recognition (INT1 and INT2)
  - It's possible to route INT1 to INT2 and vice-versa

State Machine registers

- Control registers for controlling the state machines
  - Control Register 1: for state machine 1
  - Control Register 2: for state machine 2
- Transition conditions and commands for executing the state machines algorithms
  - ST1_1 to ST1_16: state machine 1
  - ST2_1 to ST2_16: state machine 2
- Setting registers for parameters
  - 4 timers (TIM1, TIM2, TIM3, TIM4)
  - 2 thresholds (THRS1, THRS2)
- Output and status registers where to store the final result of a state machine
  - OUTS1: for state machine 1
  - OUTS2: for state machine 2

State Machine concept

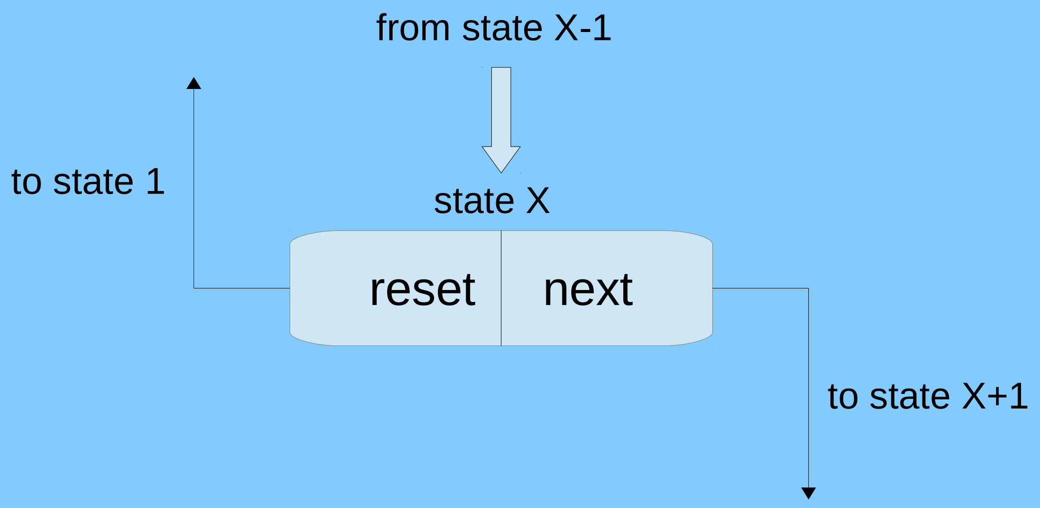
- The device supports two state machines, each state of a maximum of 16 states
- Each state has a reset and a next condition
  - The next condition brings the state machine to the next state
  - The reset condition brings the state machine to the first state
- An interrupt is generated at the final state
- A program counter points to the current state



Where n <= 16

State Machine concept

- Transition conditions are encoded on one byte to save memory size
- reset condition is stored on first 4th bits and the next on the other
  - bit from MSB to MSB-4: reset condition
  - bit from LSB to LSB+4: next condition



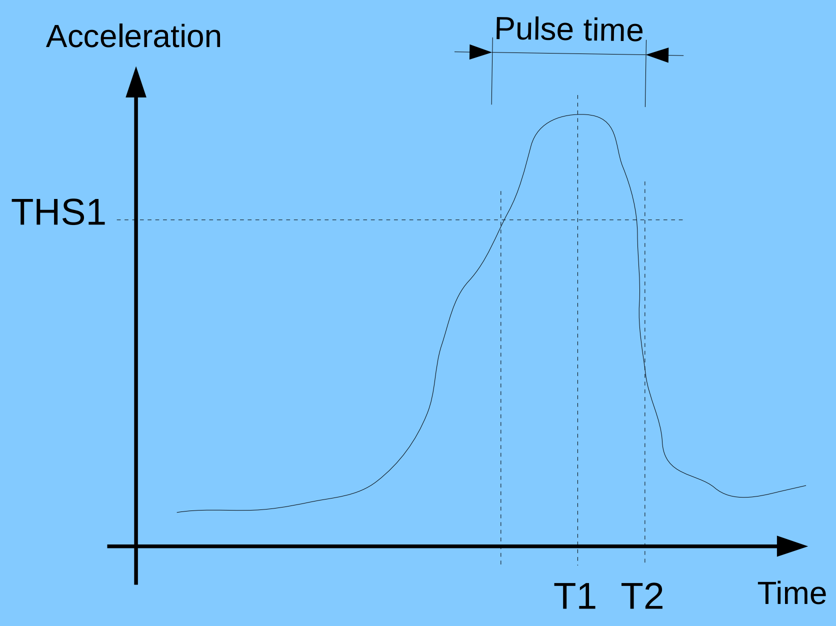
Transition condition

- Transition conditions can be used either as a reset value or as a next value

ID	**Logical name**	**Meaning**
0x0	NOP	No operation
0x1	TI1	Timer 1 valid
0x2	TI2	Timer 2 valid
0x3	TI3	Timer 3 valid
0x4	TI4	Timer 4 valid
0x5	GNTH1	Any/triggered axis greater than threshold 1
0x6	GNTH2	Any/triggered axis greater than threshold 1
0x7	LNTH1	Any/triggered axis less or equal than threshold 1
0x8	LNTH2	Any/triggered axis less or equal than threshold 2
0x9	GTTH1	All axis greater than threshold 1
0xA	LLTH2	All axis less than threshold 1
0xB	GRTH1	Any/triggered axis greater than reversed threshold 1
0xC	LRTH1	Any/triggered axis less or equal than reversed threshold 1
0xD	GRTH2	Any/triggered axis greater than reversed threshold 2
0xE	LRTH2	Any/triggered axis less or equal than reversed threshold 2
0xF	NZERO	Any axis crossing zero

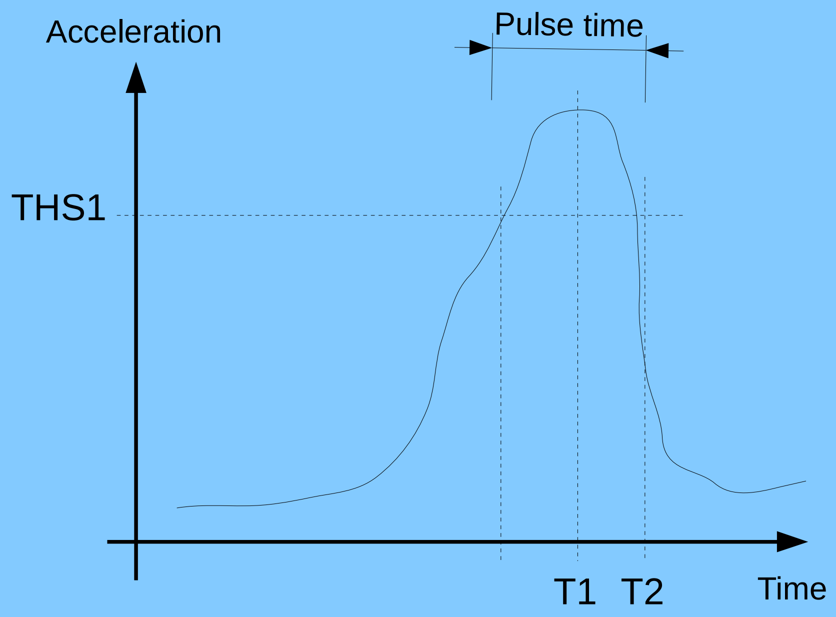
Example: pulse detection

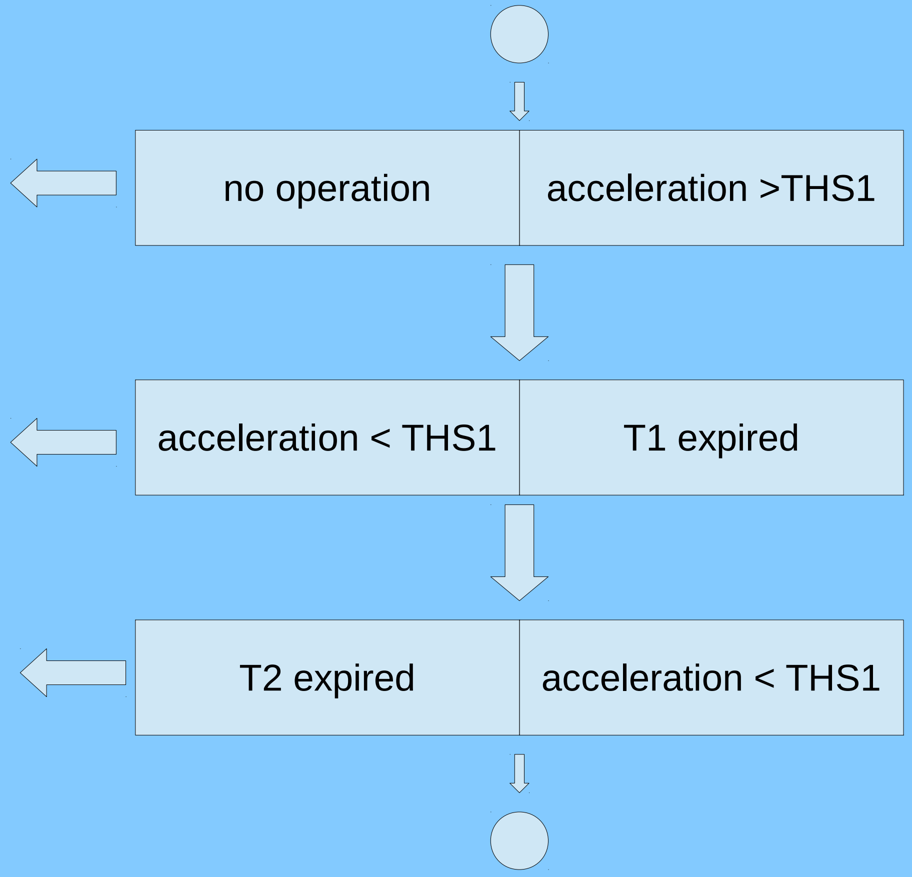
- The pulse is a short time peak of the acceleration on one axis



Example: pulse detection

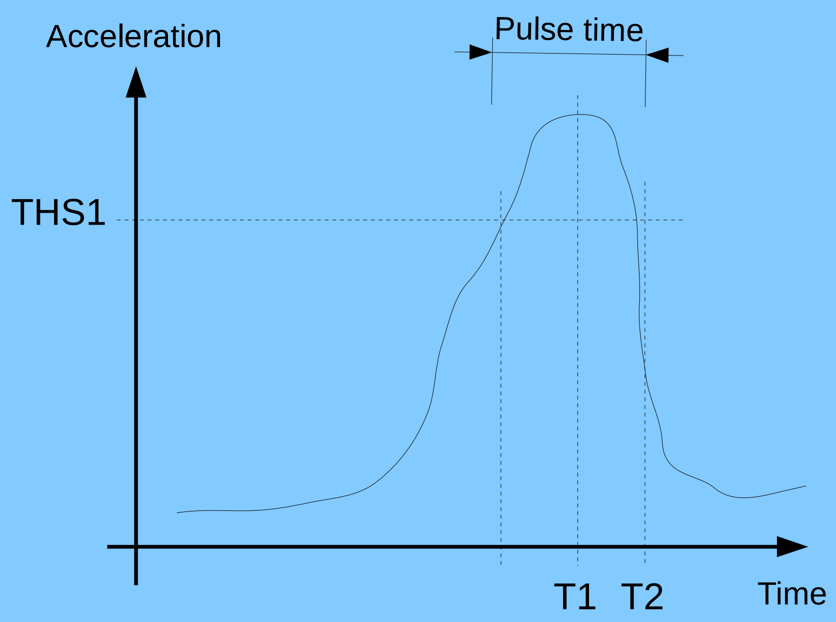
- The pulse is a short time peak of the acceleration on one axis

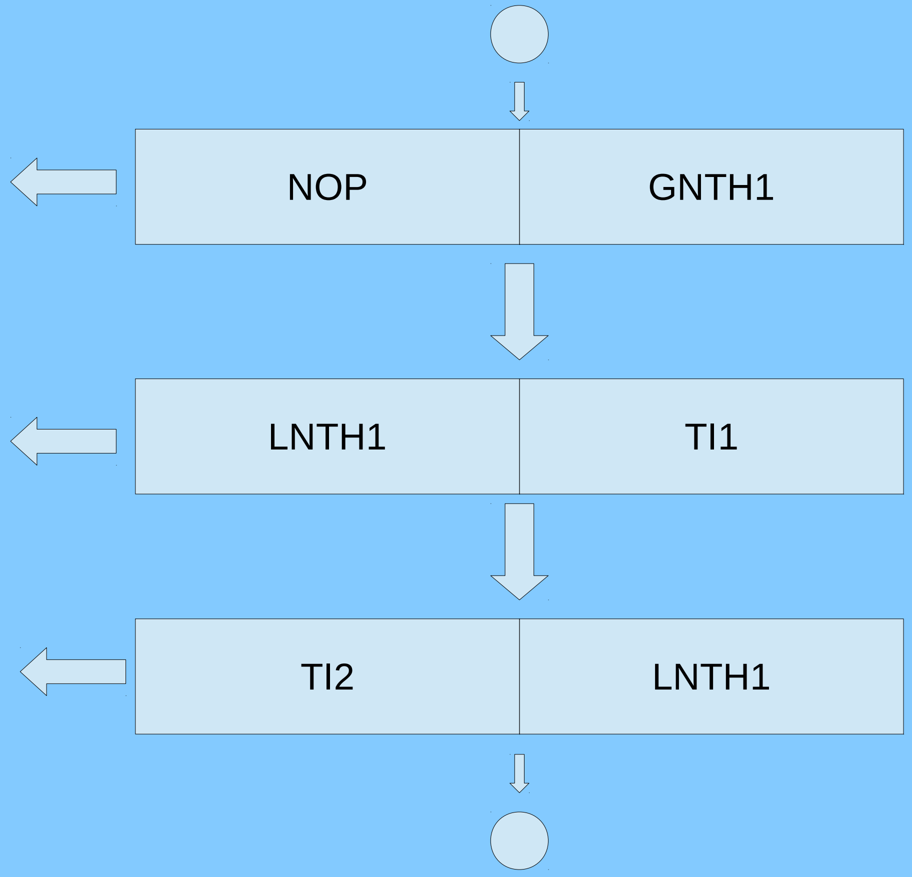




Example: pulse detection

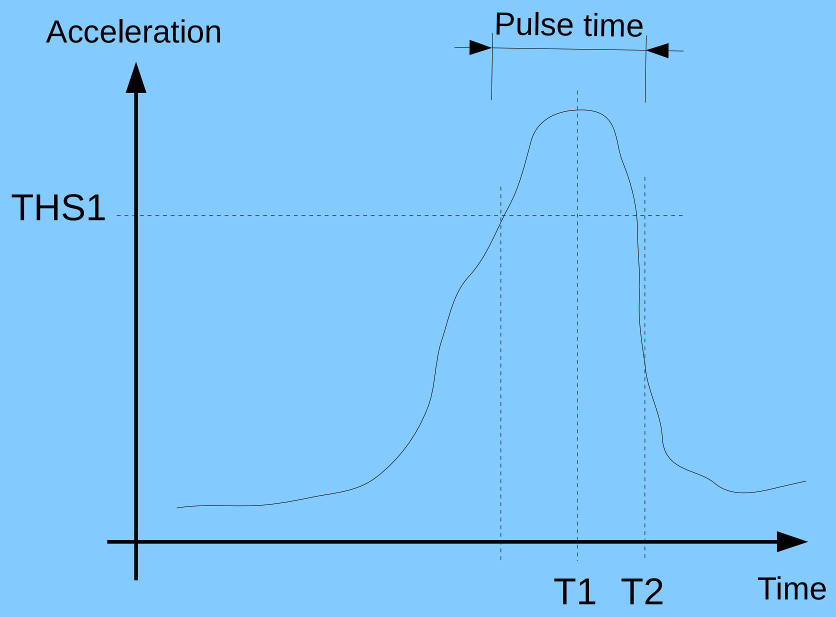
- The pulse is a short time peak of the acceleration on one axis

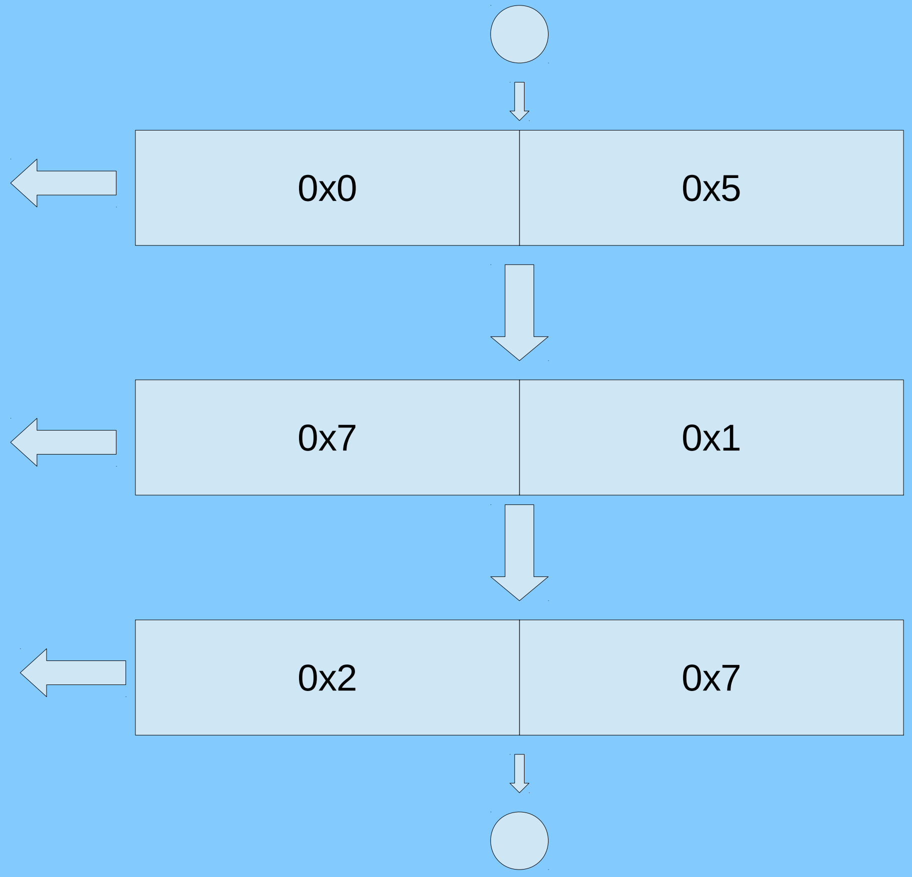




Example: pulse detection

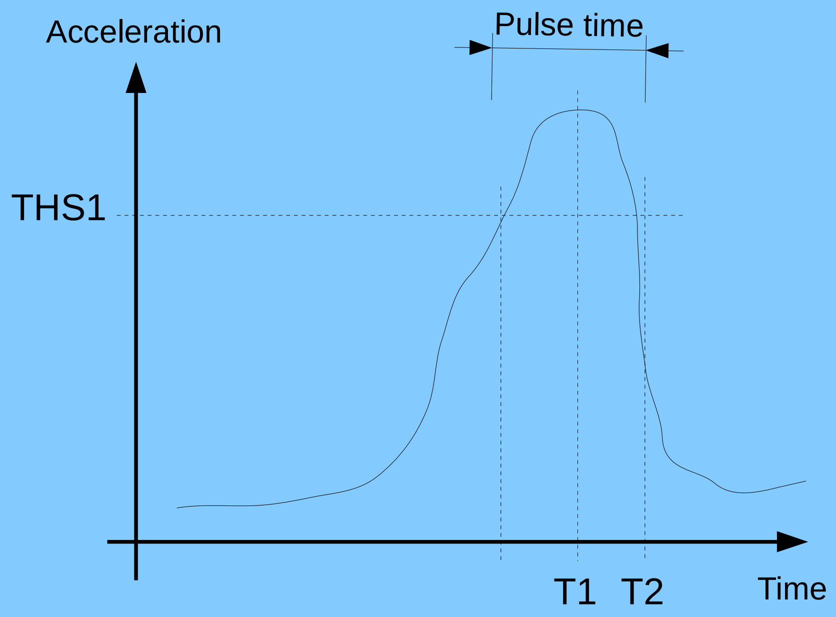
- The pulse is a short time peak of the acceleration on one axis

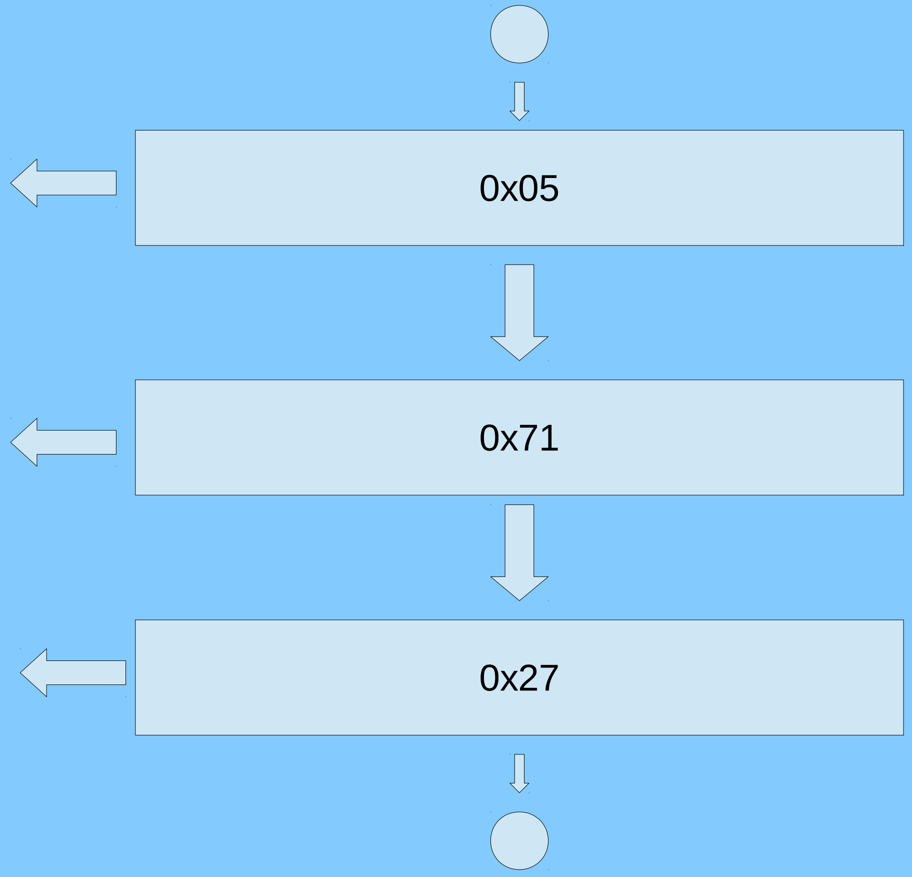




Example: pulse detection

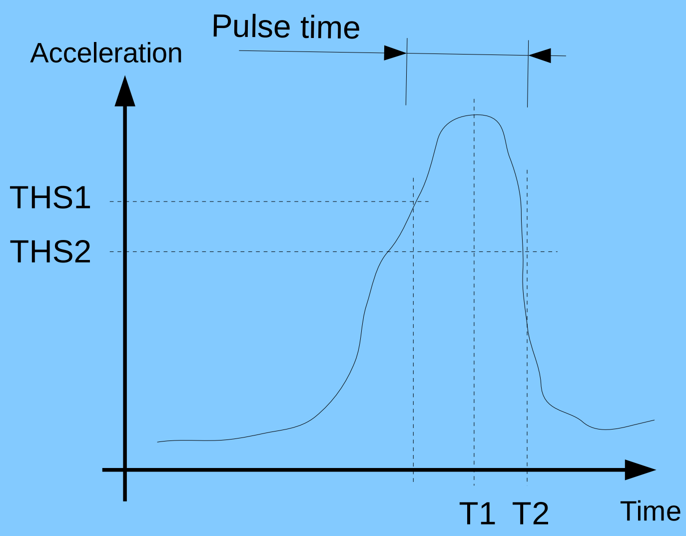
- The pulse is a short time peak of the acceleration on one axis

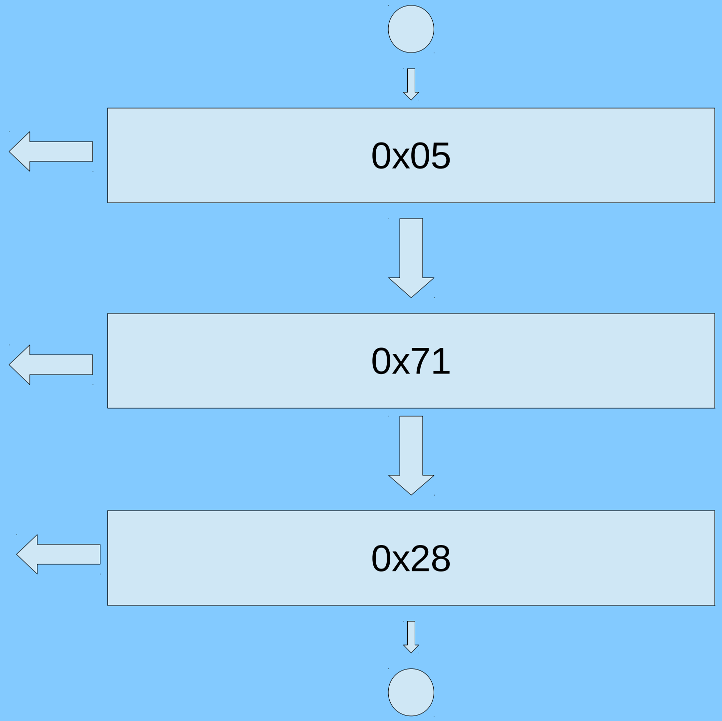




Example: pulse detection

- Pulse detection with hysteresis

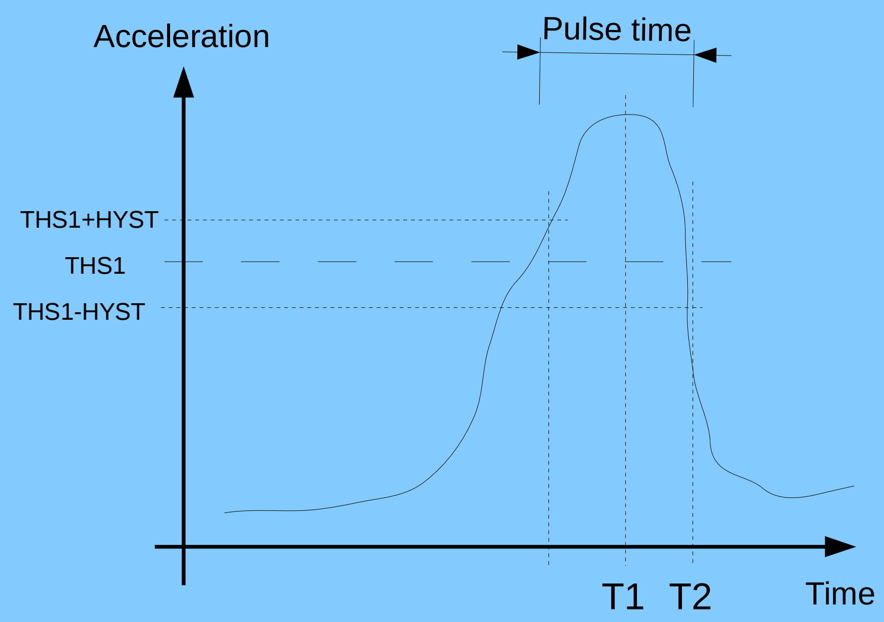


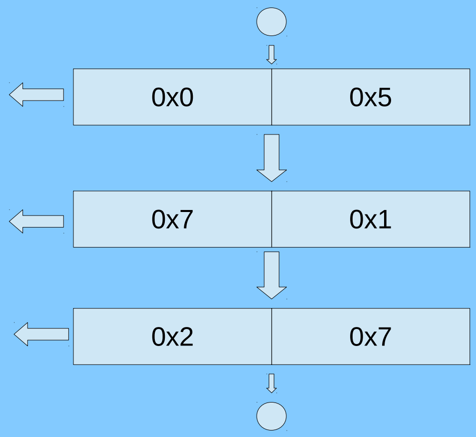


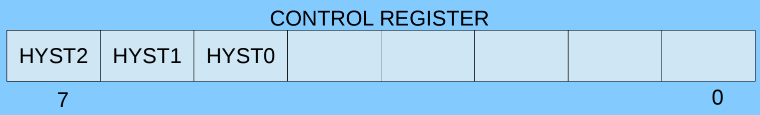
where: `0x28 = 0x2 << 4 | 0x8 = TI2 << 4 | LNTH2`

Example: pulse detection

- Pulse detection with hysteresis







GNTH1 = greater than thrs1 + hyst
LNTH1 = greater than thrs1 - hyst

Commands

- Commands allow to perform operations on the algorithm
- They are stored on the same memory where transitions are stored
- Some commands have parameters and the value is stored on the next register
- The use of commands decreases the number of states

ID	Logical name	Explanation	Parameter	Only STM2
0x00	STOP	Stop execution, and resets reset-point to	None	
0x11	CONT	Continues execution from reset-point	None	
0x22	JMP	Jump address for two Next conditions	byte 1 conditions, byte 2 jump addresses	
0x33	SRP	Set reset-point to next address / state	None	
0x44	CRP	Clear reset-point to start position (to 1st	None	
0x55	SETP	Set parameter in register memory	byte 1 address, byte 2 value	
0x66	SETS1	Set new setting to Settings 1 register	byte 1 value of settings register	
0x77	STHR1	Set new value to /THRS1_y register	byte 1 value if threshold1 register	
0x88	OUTC	Set outputs to output registers	None	
0x99	OUTW	Set outputs to output registers and wait for latch reset from host	None	
0xAA	STHR2	Set new value to /THRS2_y register	byte 1 value if threshold2 register	
0xBB	DEC	Decrease long counter -1 and validate	None	
0xCC	SISW	Swaps sign information to opposite in mask and trigger	None	
0xDD	REL	Releases temporary output information	None	
0xEE	STHR3	Set new value to /THRS3 register	byte 1 value if threshold3 register	
0xFF	SSYNC	Set synchronization point to other State	None	
0x12	SABS0	Set /SETTy, bit ABS = 0. Select unsigned filter	None	
0x13	SABS1	Set /SETTy, bit ABS = 1. Select signed filter ON	None	
0x14	SELMA	Set /MASAy pointer to MAy (set MASAy = 0)	None	
0x21	SRADI0	Set /SETT2, bit RADI = 0. Select raw data mode	None	*
0x23	SRADI1	Set /SETT2, bit RADI = 1. Select difference data mode	None	*
0x24	SELSA	Set /MASAy pointer to SAy (set MASAy = 1)	None	
0x31	SCS0	Set /SETT2, bit D_CS = 0. Select DIFF data mode	None	*
0x32	SCS1	Set /SETT2, bit D_CS = 1. Select Constant Shift data mode	None	*
0x34	STRAM0	Set /SETTy, bit R_TAM = 0. Temporary Axis Mask /TAMxAy is kept intact	None	
0x41	STIM3	Set new value to /TIM3_y register	byte 1 value if timer3 register	

Mask logic

- Mask logic is property of kxcnl/lis3dsh programmable functionality
- Doesn't affect data when streamed but only when processed by the state machine logic
- The mask allows to redefine the XYZ coordinates of the sensor
- Mask can hold information on state machines like double tap direction

Mask logic

- Mask logic is property of kxcnl/lis3dsh programmable functionality
- Doesn't affect data when streamed but only when processed by the state machine logic
- The mask allows to redefine the XYZ coordinates of the sensor
- Mask can hold information on state machines like double tap direction
- Mask calculation is done wit the formula:

$$M \times \vec{m} = \vec{m}_m$$

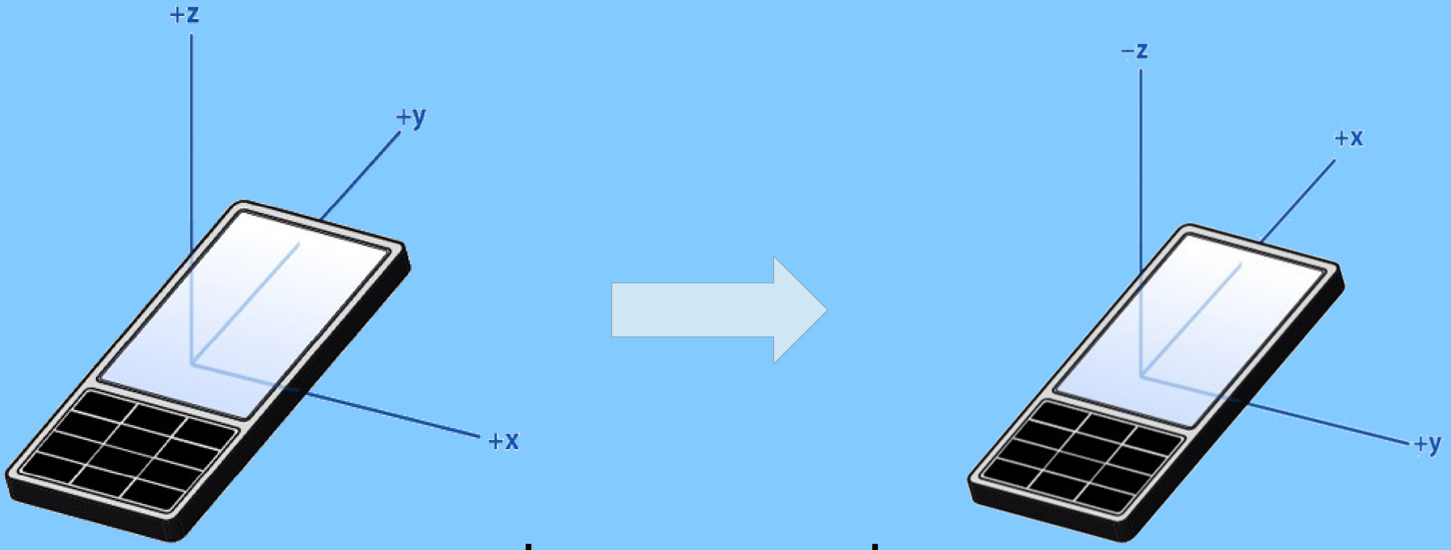
binary conversion matrix

vector containing the value to be masked

vector containing the masked value

Mask logic

Accelerometer device may be soldered on a different coordinates system



$$M = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Mask logic implementation

- Matrix building

```
/* build mask matrix */
sdata->mask_matrix[0] = 1 <<
        (ETZKX_DIMENSION - (sdata->pdata->x_map + 1) * 2 +
                                        !sdata->pdata->x_negate);
sdata->mask_matrix[1] = 1 <<
        (ETZKX_DIMENSION - (sdata->pdata->x_map + 1) * 2 +
                                        sdata->pdata->x_negate);
sdata->mask_matrix[2] = 1 <<
        (ETZKX_DIMENSION - (sdata->pdata->y_map + 1) * 2 +
                                        !sdata->pdata->y_negate);
sdata->mask_matrix[3] = 1 <<
        (ETZKX_DIMENSION - (sdata->pdata->y_map + 1) * 2 +
                                        sdata->pdata->y_negate);
sdata->mask_matrix[4] = 1 <<
        (ETZKX_DIMENSION - (sdata->pdata->z_map + 1) * 2 +
                                        !sdata->pdata->z_negate);
sdata->mask_matrix[5] = 1 <<
        (ETZKX_DIMENSION - (sdata->pdata->z_map + 1) * 2 +
                                        sdata->pdata->z_negate);
sdata->mask_matrix[6] = 2;
sdata->mask_matrix[7] = 1;
```

Mask logic implementation

- Matrix-vector remasking (multiplication)

```
static u8 etzkx_mask_orientation(struct etzkx_data *sdata, u8 val)
{
        int i;
        u8 new_val = 0;

        if (!val)
                return 0;

        for (i = 0; i < ETZKX_DIMENSION; i++)
                if (sdata->mask_matrix[i] & val)
                        new_val |= (1 << (ETZKX_DIMENSION - 1 - i));

        return new_val;
}
```

Advanced features

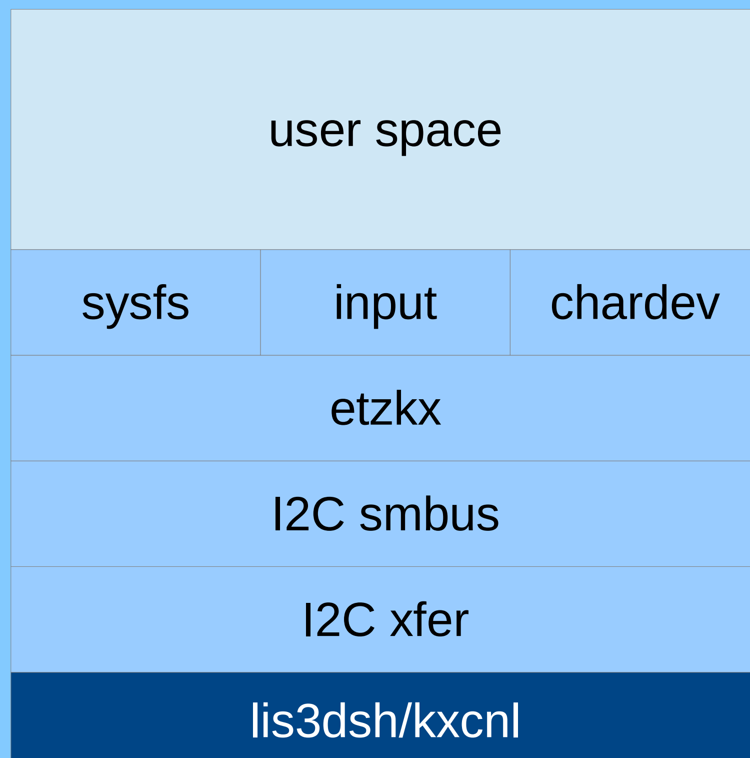
- DIFF (diff and constant shift)
- Hysteresis
- Peak detection
- Synchronized execution of state machines
- Decimation

Linux Kernel Driver

- Status of the driver
  - Soon it will be available on www.etezian.org
  - Mature enough to be sent upstream, patches are almost ready
- Location
  - drivers/misc/etzkx.c
  - include/linux/i2c/etzkx.h
  - Documentation/misc-devices/etzkx.txt
- The driver supports only Kionix kxcnl device, lis3dsh support is planned

Linux Kernel Driver

- Driver stack



Linux Kernel Driver

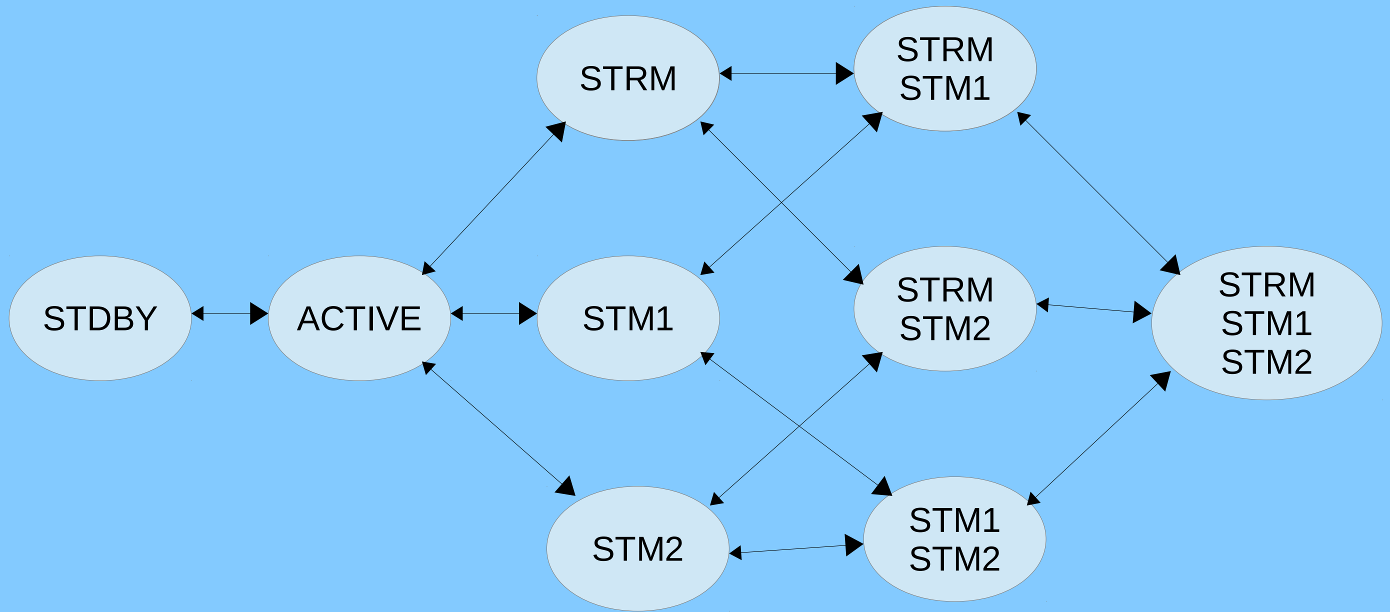
- Interfaces
  - /sys/class/i2c-adapter/i2c-n/<i2c-addr>/: handles the device
  - /dev/input/eventX: receives X, Y, Z data streaming
  - /dev/etzkx_stm: handles state machines

Linux Kernel Driver

- Character device interface (/dev/etzkx_stm)
  - Is the interface which allows to enable a specific state machine and retrieve the status of the running state machines
  - With a poll interface is possible to get the results of the enabled state machine
- Enabling/disabling state machines is done via ioctl(), awful but simplifies considerably the driver's mess

Linux Kernel Driver

- Driver state flow



Linux Kernel Driver

- State machines currently supported
  - Timing: testing state machine which jumps from a state to the next after a time threshold
  - Orientation detection: sends an interrupts every time that a change of orientation has occurred (landscape/portrait)
  - Double tap
  - Sleep/wakeup: sends an interrupt every time the device has not been moved for a time threshold and any time that the device has been moved after a sleep state

Contacts

- Contacts for hardware request
  - Rohm/Kionix: Timo Havana <timo.havana@fi.rohmeurope.com>
  - ST: Luca Fontanella <luca.fontanella@st.com>
- Contacts for software support
  - Andi Shyti <andi@etezian.org>
  - Mika Laaksonen <mika@etezian.org>
- The slides are available on
  - http://www.etezian.org/files/fosdem13_stm_accel.pdf

  and soon other related stuff will be published
- Feel free to contact me at anytime

Any questions?

Etezian.org

www.etezian.org
info@etezian.org
#etezian on freenode
http://lists.etezian.org/listinfo/etezian

Andi Shyti

Mail: andi@etezian.org
Irc: cazzacarna (freenode)
Web: www.smida.it