CA Lab3 Report

Q, Queue
Mackerels!

Contents

I Clock Implementation

I. 1 Offline Mode

On Signal lose the clock can run in offline mode and increment internal clock variable.

```
void processEventsClock(CLOCKEVENT event)
{   if (event==NOCLOCKEVENT)
        return;

    if (++secs >= 60)
    {   secs = 0;
        if (++mins >= 60)
        {   mins = 0;
            if (++hrs >= 24)
            {   hrs = 0;
                if (++day > days_in_month(month, year))
                {   day = 1;
                    if (++month > 12)
                    {   month = 1;
                        year++;
                    }
                }
                if (++weekday > 7)
                { weekday = 1;
                }
            }
        }
     }
}
```

I. 2 Changing Clock timezone

Coordinated Universal Time UTC is used to specify clock timezones.

By pressing the PTH3 button on the Dragon12 board the clock time zone will toggle DE UTC +2 summer/winter +1 timezone and US UTC -4 summer/winter -5 timezone. Depending on current timezone of the DCF77 signal

```
#define BUTTONS_POLLING_RATE   50         // once in n * 10ms
#ifdef SIMULATOR // inlined // hardware independent boolean value
  #define poll_buttons_state() (PTH)
#else
  #define poll_buttons_state() (~PTH)
#endif

#define PH3                      0x08U
#define TOGGLE_TIME_ZONE_BUTTON  PH3
static volatile void (*toggle_time_zone)(void) = toggle_de_time_zone;
void poll_buttons(void)
{
    static char counter = BUTTONS_POLLING_RATE;
    if (counter-- != 0) return;
    counter = BUTTONS_POLLING_RATE;
    if (poll_buttons_state() & TOGGLE_TIME_ZONE_BUTTON)
    {   // function pointer changes value on each call!
        toggle_time_zone(); displayTimeClock(); displayDateDcf77();
    }
}
```

I. 3 Synchronizing with External Clock

The clock keeps track of time and date separately from the DCF77 clock signal with its own time zone.

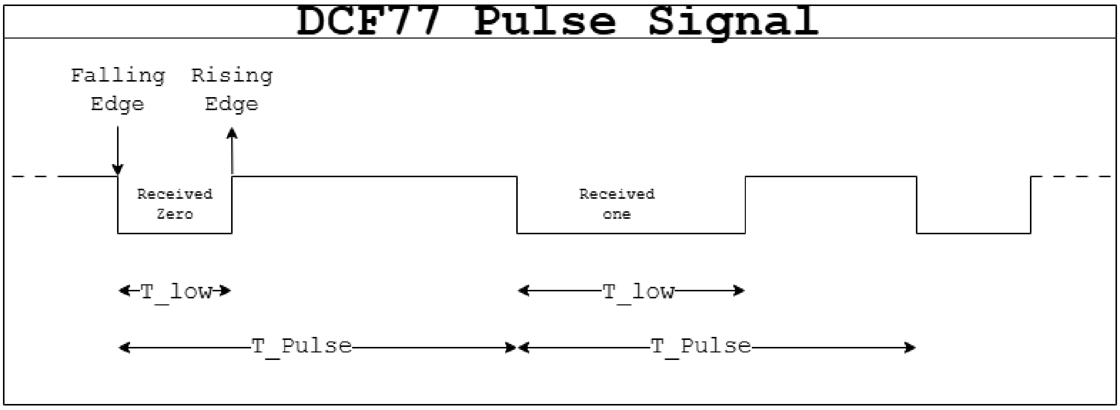
When the a valid DCF77 signal is received the setClock function is called overwrite the clock with the received clock values.

If the referenced external clock from the DCF77 signal changes the its timezone, which correspond to summer/winter time saving changes, all recorded timezone must follow that changes to.

Finally the timezone is adjusted back to the internal clock timezone.

```
void setClock(char hours, char minutes, char seconds, char _day, char _month, int
_year, char _weekday, char referenced_time_zone)
{
    char clock_time_zone, i;

    day   = _day;
    month = _month;
    year  = _year;

    hrs   = hours;
    mins  = minutes;
    secs  = seconds;

    weekday = _weekday;

    ticks = 0;

    if (referenced_time_zone != LAST_REFERENCED_CLOCK_TIME_ZONE)
    {
        // time zone change needs to stay relative to the reference
        // somer/winter time

        clock_time_zone += (referenced_time_zone - LAST_REFERENCED_CLOCK_TIME_ZONE);

        for (i = 0; i < KNOWN_TIME_ZONES_COUNT; i++)
            CURRENT_TIME_ZONES[i] +=
            (referenced_time_zone - LAST_REFERENCED_CLOCK_TIME_ZONE);

        LAST_REFERENCED_CLOCK_TIME_ZONE = referenced_time_zone;
        // not fully tested code!
    }

    clock_time_zone = CLOCK_TIME_ZONE;

    CLOCK_TIME_ZONE = referenced_time_zone;

    adjust_to_timezone(clock_time_zone);    // adjust back to current clock time zone
}
```

II DCF77 Signal Sampling



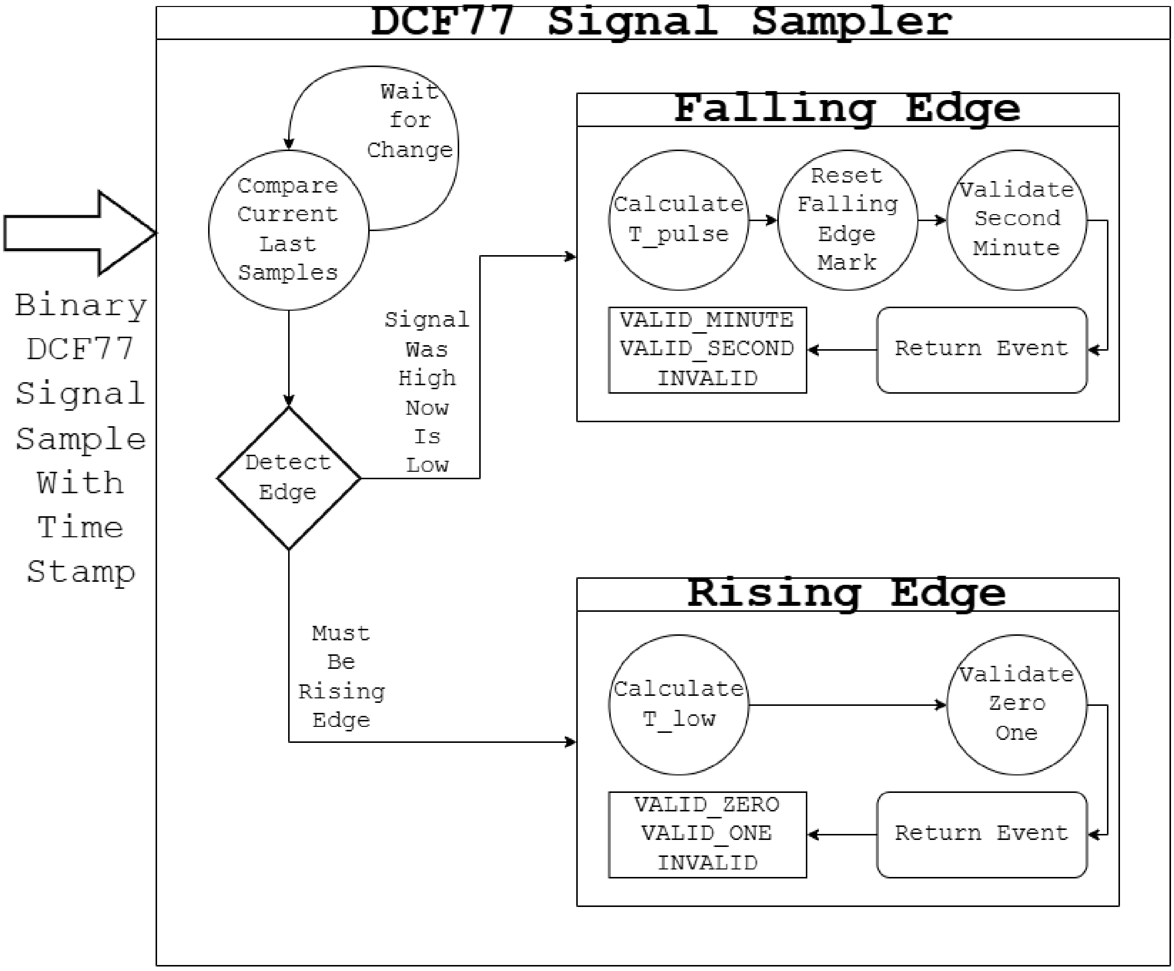


Figure 1: DCF77-Signal-Sampler

III DCF77 Signal Decoding

The signal decoding is implemented as a finite state machine using a function pointer `transition` the points to the current state of the machine.

The finite state machine has the `RESET` state defined as `waiting_for_minute_end` where if anything were to go wrong with the decoding the machine stay at that state until the `VALID_MINUTE` event comes around

```
#define RESET_FSM wait_for_minute_end

static volatile void (*transition)(DCF77EVENT event) = RESET_FSM;

void processEventsDCF77(DCF77EVENT event)
{
    switch (event)
    {
        case INVALID:
        case VALID_MINUTE:
            RESET_FSM(event);
            break;

        case VALID_ONE:
        case VALID_ZERO:
            transition(event);
            received_bit++;
            break;
    }
}
```

III. 1 Machine States

Each machine state is defined as separate function. Where the events are processed until machine transitions into next state by assigning the function pointer `transition` to the next state.

```
static void decode_minutes(DCF77EVENT event)
{
    char bit = (event == VALID_ONE) ? 1 : 0;

    if (received_bit == 28) // check parity
    {
        if (parity != bit) transition = RESET_FSM;   // invalid
        else {
            parity = 0;     // common used resource!
            transition = decode_hours;
        }
        return;
    }

    parity ^= bit;

    received_minutes += bit * TRANSMISSION_BIT_WEIGHT[received_bit - 21];
    // minutes bit weighted offset
}
```

III. 2 DCF77 Signal Decoder FSM

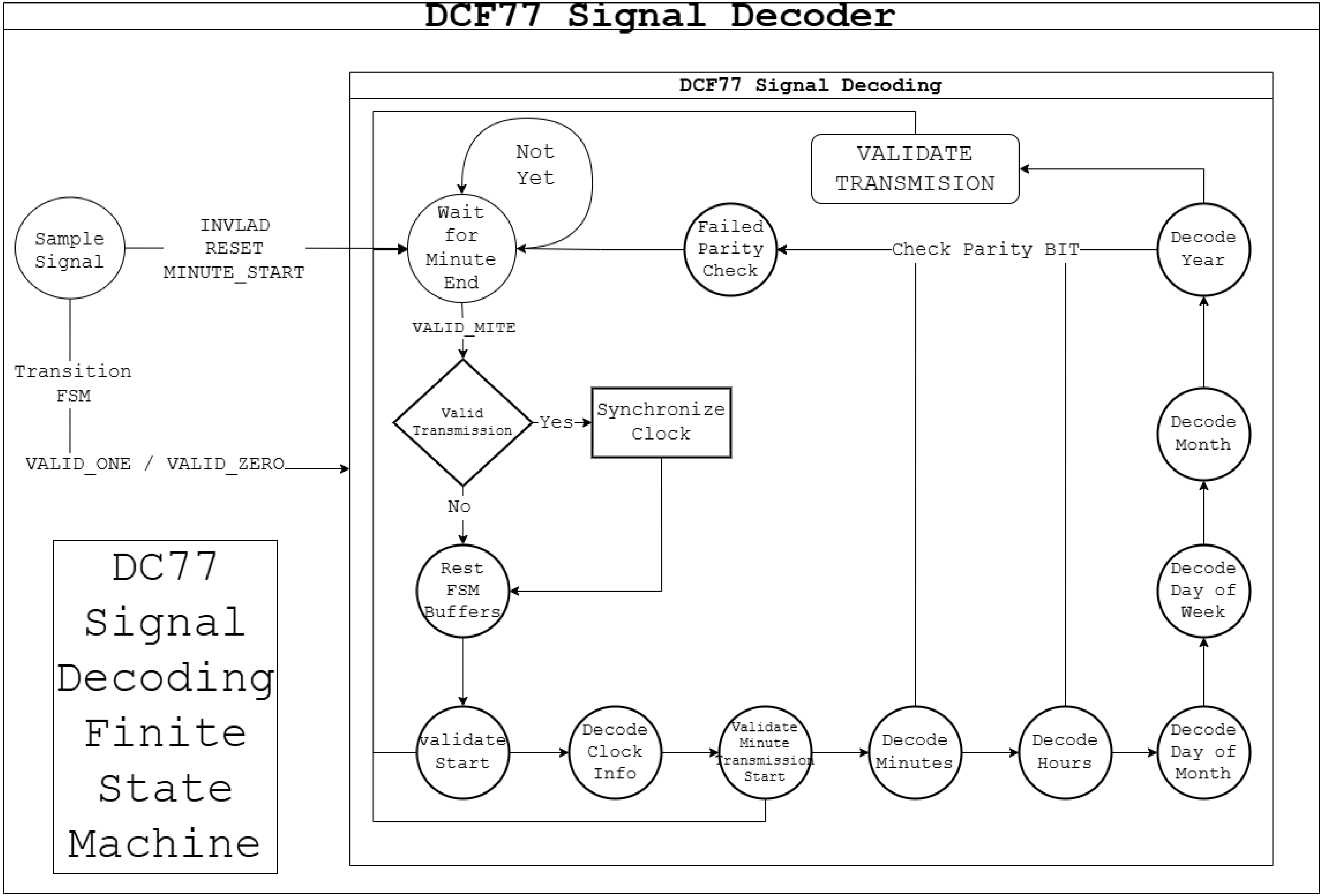


Figure 2: DCF77-Signal-Decoder-FSM