

Cleaned up by Huajian Huang on 16:36, March 11th, 2017
Author: Huajin George Huang

```
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "LOCMaster.h"
#include "ES_DeferRecall.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h" // Define PART_TM4C123GH6PM in project
#include "driverlib/gpio.h"
#include "inc/hw_timer.h"
#include "inc/hw_ssi.h"
#include "inc/hw_nvic.h"
#include "OwnPWM.h"
#include "MasterVehicle.h"

/*----- Module Defines -----*/

/*----- Module Functions -----*/

/*----- Module Variables -----*/

set up a variable to store state

set up a variable to store Priority;

set up variables to store data bytes from SPI, and important parameters extracted

set up flags to assist handshake during staging

set up variables for input capture and obtaining the frequency

define the frequency and frequency code table, and a stableCounter

define some variables to store active locations

set up some variables to store the scores

initialize a event to carry information around

/*----- Module Code -----*/
/*****
*****/
bool InitLOCMasterSM ( uint8_t Priority )
{
    save our priority
    Init SPI
    init input capture for the Hall sensor
    start the LOCMasterSM with ES_ENTRY
    start a timer for getting game status, GET_STATUS_TIMER

    return true
}
```

```

/*****
*****/
bool PostLOCMasterSM( ES_Event ThisEvent )
{
    return call post with the priority of this service
}

/*****
*****/
ES_Event RunLOCMasterSM( ES_Event CurrentEvent )
{
    default to not make a transition
    update the state
    default to normal entry to new state
    assume no error for the ReturnEvent

    switch ( CurrentState )
    {

        case WAITING :

            // This state is like a neutral transition state
            //(1) normally wait for timer and keep querying game status
            //(2) when asked to do staging area stuff, move to the corresponding states

            //In this state, I want to keep querying the game status regularly
            execute during function
            //process any events
            If an event is active
            {
                switch Event Type
                {
                    case ES_TIMEOUT:
                        If event is "The GET_STATUS_TIMER" timeout{

                            set NextState to GAME_STATUS_SENDING_TO_LOC
                            mark that we are taking a transition
                            Post the same event to self
                            consume the event
                        }
                        break;

                    case ARRIVED_AT_STAGING:

                        set NextState to SENDING_TO_LOC_AT_STAGING
                        mark that we are taking a transition
                        consume event
                        break;

                        default:
                            break;

                }
            } // end if "No event"

```

```

        else // Current Event is now ES_NO_EVENT. Correction 2/20/17 provided by
Prof.Ed
    {
        //Probably means that CurrentEvent was consumed by lower level
        return CurrentEvent as ReturnEvent
    }
    break;

    case GAME_STATUS_SENDING_TO_LOC :
        execute during function
        If an event is active
        {
            switch Event Type
            {
                case ES_TIMEOUT:
                    If event is GET_STATUS timer timeout
                    //time to query for current game status
                    write the "GET_STATUS_COMMAND" and followed by 4 bytes of 0x00
                    //Pump them into FIFO buffer, when they are all out, we get EOT interrupt
                    enable the EOT interrupt
                    set NextState to GAME_STATUS_RECEIVING_FROM_LOC
                    mark that we are taking a transition
                    consume event
                }
            }
            break;

            case ARRIVED_AT_STAGING:
                post this event back to this very service (needs to be
handled, just not now)
            break;

            default:
                break;
        }
    } //end if "no event"
    else // Current Event is now ES_NO_EVENT. Correction 2/20/17
    {
        //Probably means that CurrentEvent was consumed by lower level
        return CurrentEvent ReturnEvent
    }
    break;
// repeat state pattern as required for other states

    case GAME_STATUS_RECEIVING_FROM_LOC :
        execute during function
        If an event is active
        {
            switch Event Type
            {
                case ES_EOT: //If event is end of transmission of 5 bytes, provided by EOT
interrupt
                    if the second byte of data is 0xff (meaning this is legit data){ //all of
the useful commands start with 0xff
                        assemble the status bytes
                    }
            }
        }
    }

```

```

        //extract the bit corresponding to contruction start and see if it changes
to "construction starts"
        if the game status bit changes from "waiting to start" to "construction
active"{
            post "CONSTRUCTION_START" to MasterVehicle
        }

        if the game status bit changes from "construction active" to "waiting to
start"{
            post "CONSTRUCTION_END" to MasterVehicle
        }

        //updating current score
//depending on the team, red or green
        update the current score
        if the current score if more than the previous score{
            post a "SCORE_CHANGED" event to MasterVehicle
        }
        update previous score

        update the green and red score

        if the game is active
            //depending on the team, red or green
            update the current active stage location
            if we have an valid and active staging location and it is different from
the previous one{
                post "STAGE_ACTIVE" to MasterVehicle
            }
            update the previous active stage location

            update the current active shooting location
            if we have an valid and active shooting location and it is different
from the previous one{
                if all shooting locations are active{
                    post "SHOOT_ACTIVE_4" to MasterVehicle
                }
                else{
                    post "STAGE_ACTIVE" to MasterVehicle
                }
            }
            update the previous active shooting location

        }//end if "data is useful/legit

        update previous StatusBytes after this comparison

        start the timer for the next game status query
        set NextState to WAITING
        mark that we are taking a transition
        consume event
        break;
// repeat cases as required for relevant events

```

```

        case ARRIVED_AT_STAGING: //mainly used when we are stuck, at
we are moving around a bit and restarting
        post this event back to this very service (we need to
process it, just not now)
        break;

        default:
        break;

    }
} // end if "no event"
    else // Current Event is now ES_NO_EVENT. Correction 2/20/17
{
    //Probably means that CurrentEvent was consumed by lower level
    return CurrentEvent as ReturnEvent
}
break;

case SENDING_TO_LOC_AT_STAGING :

execute during function
If an event is active
{
    switch Event Type
    {

        case SEND_FIRST_REPORT: //if we are asked to send the first report
        clear the active location first, we are at a new round

        clear all the flags for previous report sent and ack, might be repeated, but
better safe than sorry

        if we sent a report within 200ms
        {

            set NextState to WAITING_FOR_200MS_TIMEOUT
            mark that we are taking a transition
            consume event

        }

        else if we have a freq to use that is yet to be consumed
        //write the frequency to LOC

        put that frequency in the form of the report style,
and send the whole byte in SPI

        send four 0x00 bytes following that

        enable the EOT interrupt

        raised the flag for we have sent 1st report already

        raise the flag, indicating that we have sent a report,
do not send again for another 200 ms

        start a 200 ms timer to clear this
FlagSentReprotWithin200ms

        set NextState to RECEIVING_FROM_LOC_AT_STAGING
        mark that we are taking a transitio

```

```

        consume event
    } //end the within 200ms check
    break;

    case SEND_SECOND_REPORT: //if we are asked to send the second report

        if we sent a report within 200ms
        {
            set NextState to WAITING_FOR_200MS_TIMEOUT
            mark that we are taking a transition
            consume event
        }
        else if we have a freq to use

            put that frequency in the form of the report style, and send the whole byte
            send four 0x00 bytes following that

            enable the EOT interrupt

            raised the flag for we have sent 1st report already

            raise the flag, indicating that we have sent a report, do not send again
            for another 200 ms

            start a 200 ms timer to clear this FlagSentReprotWithin200ms

            set NextState to RECEIVING_FROM_LOC_AT_STAGING
            mark that we are taking a transition
            consume event
        } //end the within 200ms check
        break;

    case GO_QUERY_REPORT_RESPONSE : //Asked to go query report response
        write the query for report response command to LOC, followed
        by 4 bytes of 0x00

        enable the EOT interrupt

        set NextState to RECEIVING_FROM_LOC_AT_STAGING
        mark that we are taking a transition
        consume event
        break;

        default:
            break;
    }

} // end if "not no-event"
else // Current Event is now ES_NO_EVENT. Correction 2/20/17
{
    //Probably means that CurrentEvent was consumed by lower level
    return CurrentEvent as ReturnEvent
}
break;

case RECEIVING_FROM_LOC_AT_STAGING :
    execute during function

```

```

//process any events
If an event is active
{
    switch Event Type
    {

        case RESTART_VERIFY_FREQ://we are stuck, restart the process
            set NextState to WAITING;
            mark that we are making a transition
            consume event
            break;

        case GO_QUERY_REPORT_RESPONSE:
            write the query for report response command to
LOC,followed by 4 bytes of 0x00
            enable the EOT interrupt
            enable the EOT interrupt

            break;

        case ES_EOT : //If event is ES_EOT, END OF FIVE BYTES, not one, FIVE!

            if we have sent the first report and not acknowledged{
                Extract to see if the response is ready
                if response is ready
                    consume the frequency once the response is ready
and our frequency is processed

                if our report is ACKed{
                    update the flags to encode our current step: sent 1st report, ACKed
1st report, not sent 2nd report, not ACKed 2nd report
                    set NextState to SENDING_TO_LOC_AT_STAGING
                    mark that we are making a transition
                    consume the event
                }
                else{ //we get NACK or Inactive, first report failed

                    update the flags to encode our current step, we go back to the
beginning, sent nothing, ACKed nothing
                    consume the frequency
                    set NextState to SENDING_TO_LOC_AT_STAGING;
                    mark that we are making a transition
                    consume the event
                }

            } // ends the "checking response ready" if statement
            else{//the response code is not ready, keep querying
                set NextState to SENDING_TO_LOC_AT_STAGING
                //keep querying until we get response ready byte
                post a "GO_QUERY_REPORT_RESPONSE" to ourself, LOCMaster
                mark that we are making a transition
                consume event
            }
        }

        //end of after first report and dealing with first ACK

        //now deal with after having one successful report

```

```

        if we sent the 2nd report and have not ACKed the 2nd report{
            //Extract to see if the response is ready
            if the response is ready{
                consume the frequency once the response is ready
            }
        }
        and our frequency is processed

        if the report is ACKed{ //data4 is RS byte, ACK is 0x00
            update the flags to encode our current step: sent 1st and 2nd report,
            ACKed 1st and 2nd report
            consume the frequency
            obtain ActiveLocation
            set NextState to WAITING;
            EventToPost.EventType=FINISHED_STAGING;
            post "FINISHED_STAGING" to MasterVehicle
            start the timer for the next game status query
        }
        otherwise it would not get triggered
        mark that we are making a transition
        consume event
    }
    else{ //we get NACK or Inactive
        clear all the sent and ACKed flags
        set NextState to SENDING_TO_LOC_AT_STAGING;
        consume the frequency
        mark that we are making a transition
        consume event
    }
}
else{//the response code is not ready, keep querying
    set NextState to SENDING_TO_LOC_AT_STAGING
    //keep querying until we get response ready byte
    post a "GO_QUERY_REPORT_RESPONSE" to ourself, LOCMaster
    mark that we are making a transition
    consume event
}
} //end of after first report and dealing with first ACK

break;
// repeat cases as required for relevant events
default:
    break;
}
} // end if "no event"
else // Current Event is now ES_NO_EVENT. Correction 2/20/17
{
    //Probably means that CurrentEvent was consumed by lower level
    return CurrentEvent as ReturnEvent
}
break;

case WAITING_FOR_200MS_TIMEOUT :
    execute during function

//process any events
If an event is active
{
    switch Event Type
    {

        case RESTART_VERIFY_FREQ://restart when we are stuck
    }
}

```



```

        set NextState to WAITING;
        mark that we are making a transition
    consume event
        break;

    case ES_TIMEOUT :
        If event is the 200ms timeout{
            clear the flag that we went a report within 200ms
            if we failed the previous handshake and all sent/ACKed
flags are low{//failed on any attempt, restarting

            set NextState = SENDING_TO_LOC_AT_STAGING
            consume the frequency and let input capture post
            mark that we are taking a transition
            consume event
                }

            else if we sent and ACKed 1st report{//sent one report,
ACKed first report, sent to this state when attempting to send the second report

            set NextState to SENDING_TO_LOC_AT_STAGING
            consume the frequency and let input capture post
            mark that we are taking a transition
            consume event
                }

            }//end checking the timeout is from the 200ms report timer
            break; //break the timeout case

        default:
            break;//break default, for switching event type
        }//end switch event type

    } //end if "not no event"
    else // Current Event is now ES_NO_EVENT. Correction 2/20/17
    {
        //Probably means that CurrentEvent was consumed by lower level
        return CurrentEvent as ReturnEvent // in that case update ReturnEvent too.
    }

    break; //break the waiting 200ms state

    default:
        break;//break default, for switching states
    } //end switch of states

If we are making a state transition
{
    Execute exit function for current state
    Modify state variable
    Execute entry function for new state

}

return(ReturnEvent);
}
/*****
*****/

```

```

void StartLOCMasterSM ( ES_Event CurrentEvent )
{

    set our initial state is WAITING
    run the state machine

    return true;
}

/*****
private functions
*****/

static ES_Event DuringWaitingState( ES_Event Event)
{
    assume no re-mapping or consumption for the ReturnEvent

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if we have ES_ENTRY or ES_ENTRY_HISTORY
    {
        set all the flags related to handshaking at staging area to default/initial value
    }
    else if we have ES_EXIT
    {}
    else
    {}
    return ReturnEvent
}

static ES_Event During_GAME_STATUS_SENDING_TO_LOC_State( ES_Event Event)
{
    assume no re-mapping or consumption for ReturnEvent

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if we have ES_ENTRY or ES_ENTRY_HISTORY
    {}
    else if we have ES_EXIT
    {}
    else
    {}

    return ReturnEvent
}

static ES_Event During_GAME_STATUS_RECEIVING_FROM_LOC_State( ES_Event Event)
{
    assume no re-mapping or consumption for ReturnEvent

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if we have ES_ENTRY or ES_ENTRY_HISTORY
    {}
    else if we have ES_EXIT
    {}
    else
    {}
}

```

```

    return ReturnEvent
}

static ES_Event During_SENDING_TO_LOC_AT_STAGING_State( ES_Event Event)
{
    assume no re-mapping or consumption for ReturnEvent

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if we have ES_ENTRY or ES_ENTRY_HISTORY
    {}
    else if we have ES_EXIT
    {}
    else
    {}

    return ReturnEvent
}

static ES_Event During_RECEIVING_FROM_LOC_AT_STAGING_State( ES_Event Event)
{
    assume no re-mapping or consumption for ReturnEvent

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if we have ES_ENTRY or ES_ENTRY_HISTORY
    {}
    else if we have ES_EXIT
    {}
    else
    {}

    return ReturnEvent
}

static ES_Event DuringWait200msState( ES_Event Event)
{
    assume no re-mapping or consumption for ReturnEvent

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if we have ES_ENTRY or ES_ENTRY_HISTORY
    {}
    else if we have ES_EXIT
    {}
    else
    {}

    return ReturnEvent
}
/*****SPI
FUNCTIONS*****/

void SPI_Init(void){
    Enable the clock to the GPIO Port (we are going to use Port A)
    Enable clock to SSI - set to SSI Module 0
    Wait for GPIO Port to be ready by killing a few cycles

    Program the GPIO to use the alternate functions on the SSI pins PA2,3,4,5
    Set Mux position in GPIOCTL to select the SSI use of the pins
    Program the port lines for digital I/O
    Program the required data directions on the port line

```

```

    program the pull-up on the clock line
    Wait for the SSI0 to be ready
    Make sure that the SSI is disabled before programming mode bits
    select Master mode and TXRES indicating EOT
    Configure the SSI clock source to the system clock
    Configure the clock pre-scaler: here we want CPSDVSR = 80 , 1+SCR = 61
    Configure clock rate (SCR) - 0, phase (SPH)- 1 and polarity (SP0)- 1 ,
    mode (FRF) - freescale(0) and datasize (DSS) - 8 bit

    Locally Enable Interrupts (TXIM in SSIIM)
    Enable SSI
    Globally enable interrupts
    enable SSI3 interrupt in the NVIC, it is interrupt number 7 so appears in EN0 at bit 7
    make sure we disable loopback mode
}

void SPI_Interrupt_Response(void){\

    disable the interrupt

    consecutive five reads from FIFO
    post "ES_EOT" to LOCMaster //it's a 5-byte E.O.T.
}

/*****Other functions*****/
uint32_t QueryGameStatus(void){

    return StatusBytes, which contains SB1 to SB4
}

uint8_t QueryActiveLocation(void){

    return ActiveLocation
}

/*****Input Capture related to Hall Sensor*****/
void InitInputCapture_Hall( void ){
    start by enabling the clock to the timer (Wide Timer 5)
    enable the clock to Port D
    // since we added this Port D clock init, we can immediately start
    // into configuring the timer, no need for further delay
    make sure that timer (Timer A) is disabled before configuring
    set it up in 32bit wide// (individual, not concatenated) mode
    the constant name derives from the 16/32 bit timer, but this is a 32/64
    bit timer so we are setting the 32bit mode
    register to 0xffff.ffff (its default value)

    set up timer A in capture mode (TAMR=3, TAAMS = 0),
    for edge time (TACMR = 1) and up-counting (TACDIR = 1)

    To set the event to rising edge, we need to modify the TAEVENT bits
    in GPTMCTL. Rising edge = 00, so we clear the TAEVENT bits

    Now Set up the port to do the capture (clock was enabled earlier)

```

```

    start by setting the alternate function for Port D bit 6 (WT5CCP0)

    map bit 6 alternate function to WT5CCP0
    // 7 is the mux value to select WT0CCP0, 16 to shift it over to the
    // right nibble for bit 6 (4 bits/nibble * 6 bits)

    Enable pin on Port D for digital I/O
make pin 4 on Port D into an input
    back to the timer to enable a local capture interrupt
    enable the Timer A in Wide Timer 0 interrupt in the NVIC
    // it is interrupt number 104 so appears in EN3 at bit 8
    make sure interrupts are enabled globally
    now kick the timer off by enabling it and enabling the timer to
    stall while stopped by the debugger
}

void InputCaptureResponse_Hall( void ){
    ES_Event EventToPost_InputCapture;

    start by clearing the source of the interrupt, the input capture event
    now grab the captured value and calculate the period
    update LastCapture to prepare for the next edge

    calculate what frequency code we have for this period we measured

        if it is the same as the previous one{
            increment stableCounter
        }
        else{
            restart the count, stableCounter
        }

    if we have read enough times of the same measurement, we know it is stable{
        update the stableMeasFreqCode with the current measFreqCode
        restart the count
    }
    //only post frequency when we have a valid and stable frequency code
    if we are in a state dealing with staging{
        //decide whether that's a code for 1st report or 2nd report
        if we have not sent the 1st report{
            raise flag indicating we have a valid frequency to process
            post "SEND_FIRST_REPORT" with the stable measured frequency code to LOCMaster
        }
        else if we have not sent the 2nd report and 1st report is ACKed{
            raise flag indicating we have a valid frequency to process
            post "SEND_SECOND_REPORT" with the stable measured frequency code to LOCMaster
        }
    }
}

    update prevMeasFreqCode

} //end input capture response

uint8_t frequency_map(uint32_t period){

```

```

    //input is in encoder ticks
    uint8_t result, assume we have no valid frequency to start with
    //the input would be in ticks, 4*10^7 ticks in one sec, period table is in micro seconds,
    //4*10^7 ticks in one sec is 4*10^7 ticks in 10^6 micro
    calculate the period in micro seconds

    loop through the frequency table to find the frequency code

    return result;
}

//-----functions used to interact with other modules-----
uint32_t queryStatusBytes(void){
    return StatusBytes;
}

uint8_t queryActiveStagingGreen(void){
    check whether the game is on or not, whether we are staging or not, and obtain the active
    location from status bytes
    returns 0 for no active staging (either game has not started or in shooting), 1, 2, 3 for
    1R/1G, 2R/2G, 3R/3G, 5 means error

    return ReturnResult;
}

uint8_t queryActiveShootingGreen(void){

    check whether the game is on or not, whether we are shooting or not, and obtain the
    active location from status bytes
    returns 0 for no active staging (either game has not started or in shooting), 1, 2, 3 for
    1R/1G, 2R/2G, 3R/3G,
    4 means all goals are open, 5 means error

    return ReturnResult;
}

uint8_t quickQueryActiveLocation(void){
    //instead of waiting for the GAME_STATUS to update, just grab it from the response
    return ActiveLocation;
}

uint8_t queryGoalGreen(void){
    return (StatusBytes & GOAL_SCORE_GREEN_MASK) >> GOAL_SCORE_GREEN_OFFSET;
}

uint8_t queryGoalRed(void){
    return StatusBytes & GOAL_SCORE_RED_MASK;
}

void check_active_event(){
    //depends on team color

    // Check Staging
    update the current active staging location

    if we have an active valid location{

```

```
        post "STAGE_ACTIVE" to MasterVehicle
    }
    update the previous active staging location

    // Check shooting
    update the current active shooting location

    if we have an active valid location{
        post "SHOOT_ACTIVE" to MasterVehicle
    }
    update the previous active shooting location
}
```