Reading Assignment

- This lecture: 5.5, 5.6
- Next lecture: 5.7 – 5.9
Outline

Software Synthesis Overview

Communication Synthesis
Software synthesis: generate binary code for target platform
Software Synthesis Flow

FIGURE 5.2 Software synthesis flow (Gajski et al.)
Code Generation

- Input: a single task specified in certain system design language, e.g. SystemC
- Output: a sequential program in a programming language that will be compiled to binaries later, e.g. C
- Translate module compositions and communications into available language constructs.
  - Use data abstractions to represent modules, ports, and their compositions
  - Use function call and global variables to resolve communications within the same task.
  - Other communications are delegated to lower levels.
Multi-Task Synthesis

- For dynamic on-line scheduling
- RTOS-based multi-tasking
  - User tasks are executed on top of an off-the-shelf RTOS and are scheduled by the RTOS scheduler.
  - Task management and synchronization are resolved by the RTOS Abstraction Layer (RAL) API that hides OS details.
  - Preferred when there is enough resource due to its flexibility and maturity
- Interrupt-based multi-tasking
  - Applicable when off-the-shelf RTOS’ are not suitable due to performance and resource constraints.
  - Also similar to how RTOS implements task management and task scheduling.
Interrupt-Based Multi-Tasking

- Specify tasks as FSMs explicitly.
  - Tasks are suspended/resumed after/before state transitions.
  - The states should encode everything including processor state and the stack.
- The task should be reorganized so the states can be introduced at strategic locations to save storage requirement.
  - e.g. it is not wise to introduce a state deep inside a function call hierarchy.
  - Memory constraints may demand to share a single stack among multiple tasks.
  - Executing times for state transitions are also constrained to achieve certain scheduling goals.
Interrupt-Based Multi-Tasking

- Specify tasks as FSMs explicitly.
  - Tasks are suspended/resumed after/before state transitions.
  - The states should encode everything including processor state and the stack.
- The task should be reorganized so the states can be introduced at strategic locations to save storage requirement.
  - e.g. it is not wise to introduce a state deep inside a function call hierarchy.
  - Memory constraints may demand to share a single stack among multiple tasks.
  - Executing times for state transitions are also constrained to achieve certain scheduling goals.
Example FSM Generation

(a) Input

(b) Output

**FIGURE 5.9** Interrupt-based multi-tasking example
Interrupts-Based C Multi-Tasking

1 /* interrupt handler */
2 void intHandler_I1() {
3   release(S1); /* set S1 ready */
4   executeTask0(); /* task state machine */
5 }
6 /* task state machine */
7 void executeTask0() {
8   do { switch(Task0.State) {
9     /* ... */
10       case ST1: C1(...);
11         Task0.State = ST2;
12       case ST2: if(attempt(S1)) T1_receive(...);
13         else break;
14         C2(...);
15         Task0.State = ST3;
16       case ST3: /* ... */
17       } } while (Task0.State == ST1);
18 }

FIGURE 5.9
LISTING 5.5 State machine implementation

(Gajski et al.)
Outline

Software Synthesis Overview

Communication Synthesis
Inter-Process Communications (IPC)

- Communications between tasks on the same processor
- Implement abstract channels utilizing synchronization primitives provided by RTOS

*(Gajski et al.)*

*FIGURE 5.10* Internal communication
Communications between tasks on the same processor
Implement abstract channels utilizing synchronization primitives provided by RTOS
Example: Handshake Channel

```c
/** SHS OS-specific struct */
typedef struct {
    sem_t req; /**< os semaphore */
} tESE_ch_shs;

void ESE_shs_init(tESE_ch_shs *pThis) {
    int retVal = sem_init(&pThis->req, 0, 0);
    /* ... error handling */
}

void ESE_shs_send(tESE_ch_shs *pThis) {
    int retVal = sem_post(&pThis->req);
    /* ... error handling */
}

void ESE_shs_receive(tESE_ch_shs *pThis) {
    int retVal = sem_wait(&pThis->req);
    /* ... error handling */
}
```

LISTING 5.6 Internal communication example of single handshake (Gajski et al.)

▶ What if we need to pass a message across the channel?
Example: Handshake Channel

```c
/** SHS OS—specific struct */
typedef struct {
    sem_t req; /* < os semaphore */
} tESE_ch_shs;

void ESE_shs_init(tESE_ch_shs *pThis){
    int retVal = sem_init(&pThis->req, 0, 0);
    /* ... error handling */
}

void ESE_shs_send(tESE_ch_shs *pThis){
    int retVal = sem_post(&pThis->req);
    /* ... error handling */
}

void ESE_shs_receive(tESE_ch_shs *pThis){
    int retVal = sem_wait(&pThis->req);
    /* ... error handling */
}
```

LISTING 5.6 Internal communication example of single handshake (Gajski et al.)

- What if we need to pass a message across the channel?
Communications between tasks on different processors
- Implement various drivers to support communications with external components/tasks.
  - It is reasonable to follow the 7-layer model.
Communications between tasks on different processors
Implement various drivers to support communications with external components/tasks.
  - It is reasonable to follow the 7-layer model.
Use marshalling (and demarshalling) to create (and to retrieve data from) processor independent layout

- Host-to-network functions take care of byte endianness.
- Marshalling/demarshalling code could be generated automatically.

What if a more complicated data structure should be supported?
Use marshalling (and demarshalling) to create (and to retrieve data from) processor independent layout

- Host-to-network functions take care of byte endianness.
- Marshalling/demarshalling code could be generated automatically.

What if a more complicated data structure should be supported?

```c
typedef struct tReq {
    long startTime;
    short coeff1;
    unsigned short base;
} tReq;

LISTING 5.7 User type definition

void myCh_send(/*_...*/ This, struct tReq *pD){
    unsigned char *pB = This->buf;
    htonlong(pB, pD->startTime);
    pB += 4;
    htonlshort(pB, pD->coeff1);
    pB += 2;
    htonlushort(pB, pD->base);
    pB += 2;
    DLink0_trans_send(/*_...*/ This->buf, 8);
}

LISTING 5.8 Marshalling code

FIGURE 5.12 Marshalling example
```
Utilize a low level service that supports only messages with fixed lengths via packetization.

What if multiplexing is necessary?
Utilize a low level service that supports only messages with fixed lengths via packetization.

What if multiplexing is necessary?
Figure 5.14 Chain for interrupt-based synchronization
(Gajski et al.)
Events in Interrupt-Based Synchronization

Each driver component handles a set of events.

- Hardware ISR preempts the current task.
- SysInt queries PIC for detailed interrupt data.
- INTC queries hardware status and triggers UsrInt2 that posts the semaphore.

*(Gajski et al.)*
FIGURE 5.16 Polling-based synchronization

(Gajski et al.)
Events in Polling-Based Synchronization

- The task repeatedly checks hardware status and yields if cannot proceed.
- How often should the task check?
  - More often: overhead in scheduling, waste processor cycles
  - Less often: the hardware may need to wait when ready

**Figure 5.17** Events in polling-based synchronization (Gajski et al.)
Events in Polling-Based Synchronization

The task repeatedly checks hardware status and yields if cannot proceed.

How often should the task check?
- More often: overhead in scheduling, waste processor cycles
- Less often: the hardware may need to wait when ready
MAC Layer

**FIGURE 5.18** Transferring a packet using bus primitives

```c
void masterWrite(unsigned int addr, void *pD, unsigned int len) {
    unsigned char *p = (unsigned char*)data;
    while (len >= 4) {
        *((unsigned int*)addr) = *((unsigned int*)pD);
        len -= 4; pD += 4;
    }
    if (len >= 2) { /* remaining short */
        *((unsigned short*)addr) = *((unsigned short*)pD);
        len -= 2; pD += 2;
    }
    if (len >= 1) { /* the last byte */
        *((unsigned char*)addr) = *((unsigned char*)pD);
        len -= 1; pD += 1;
    }
}
```

**LISTING 5.10** MAC driver example

- e.g. via a memory-mapped I/O

(Gajski et al.)
Summary

- Internal communications are built on top of IPC primitives provided by RTOS.
- Drivers for external communications are separated into layers.